KUROSE | ROSS

Redes de computadores e a internet

uma abordagem top-down

6ª edição

ALWAYS LEARNING

PEARSON

Redes de computadores e a internet

KUROSE | ROSS

Redes de computadores e a internet

uma abordagem top-down

6ª edição

SOBRE OS AUTORES

Jim Kurose

Jim Kurose é um destacado professor universitário de Ciência da Computação na Universidade de Massachusetts, Amherst.

Dr. Kurose recebeu diversos reconhecimentos por suas atividades educacionais, incluindo o Outstanding Teacher Awards da National Technological University (oito vezes), na Universidade de Massachusetts e na Northeast Association of Graduate Schools. Recebeu a IEEE Taylor Booth Education Medal e foi reconhecido por sua liderança da Commonwealth Information Technology Initiative de Massachusetts. Também recebeu um GE Fellowship, um IBM Faculty Development Award e um Lilly Teaching Fellowship.

Foi editor-chefe da *IEEE Transactions of Communications* e da IEEE/ACM Transactions on Networking. Trabalhou ativamente nos comitês de programa para *IEEE Infocom*, *ACM SIGCOMM*, *ACM Internet Measurement Conference* e *ACM SIGMETRICS* por vários anos, e atendeu como copresidente de programa técnico nessas conferências. Ele é *fellow* do IEEE e da ACM. Seus interesses de pesquisa incluem protocolos e arquitetura de rede, medição de redes, redes de sensores, comunicação multimídia e modelagem e avaliação de desempenho. Tem doutorado em Ciência da Computação pela Universidade de Columbia.

Keith Ross

Keith Ross é professor na cátedra de Leonard J. Shustek e diretor do Departamento de Ciência da Computação no Instituto Politécnico da Universidade de Nova York. Antes de ingressar nesse Instituto em 2003, foi professor na Universidade da Pensilvânia (13 anos) e no Eurécom Institute (5 anos). Obteve bacharelado pela Universidade Tufts, mestrado pela Universidade de Columbia e doutorado em Computador e Engenharia de Controle pela Universidade de Michigan. Keith Ross também é fundador e CEO original da Wimba, que desenvolve aplicações de multimídia *on-line* para *e-leaming* e foi adquirida pela Blackboard em 2010.

Os interesses de pesquisa do professor Ross estão em segurança e privacidade, redes sociais, redes P2P, medição na Internet, fluxo contínuo de vídeo, redes de distribuição de conteúdo e modelagem estocástica. É *fellow* do IEEE, recebeu o Infocom 2009 Best Paper Award e também os Best Paper Awards de 2011 e 2008 por Comunicações em Multimídia (concedido pela IEEE Communications Society). Trabalhou em diversos comitês editoriais de jornal e comitês de programa de conferência, incluindo IEEE/ACM Transactions on Networking, ACM SIGCOMM, ACM CoNext e ACM Internet Measurement Conference. Ele também trabalhou como consultor de compartilhamento de arquivos P2P para a Federal Trade Commission.

Para Julie e nossas três preciosidades:
Chris, Charlie e Nina
JFK

Um grande MUITO OBRIGADO aos meus professores, colegas
e alunos do mundo inteiro.
KWR

PREFÁCIO

Bem-vindo à sexta edição de *Redes de computadores e a Internet: uma abordagem top-down*. Desde a publicação da primeira edição, há doze anos, nosso livro foi adotado em centenas de universidades e instituições de ensino superior, traduzido para mais de 14 idiomas e utilizado por mais de cem mil estudantes e profissionais no mundo inteiro. Muitos desses leitores entraram em contato conosco e ficamos extremamente satisfeitos com sua reação positiva.

QUAIS SÃO AS NOVIDADES DA SEXTA EDIÇÃO?

Acreditamos que uma importante razão para esse sucesso é que o livro continua a oferecer uma abordagem moderna do ensino de redes de computadores. Fizemos mudanças nesta sexta edição, mas também mantivemos inalterado o que acreditamos (e os instrutores e estudantes que usaram nosso livro confirmaram) serem os aspectos mais importantes do livro: sua abordagem *top-down*, seu foco na Internet e um tratamento moderno das redes de computadores, sua atenção aos princípios e à prática, e seu estilo e método acessíveis em relação ao aprendizado de redes de computadores. Apesar disso, a esta edição foi revisada e atualizada de modo substancial:

- O site de apoio do livro foi significativamente expandido e enriquecido para incluir exercícios interativos, conforme discutido mais adiante neste Prefácio.
- No Capítulo 1, o tratamento das redes de acesso foi modernizado e a descrição do ecossistema de ISP da Internet foi substancialmente revisada, considerando o surgimento recente das redes de provedor de conteúdo, como a do Google. A apresentação da comutação de pacotes e da comutação de circuitos também foi reorganizada, oferecendo uma orientação mais tópica, em vez de histórica.
- No Capítulo 2, Python substituiu Java para a apresentação da programação de *sockets*. Embora ainda expondo explicitamente as ideias por trás da API *sockets*, o código Python é mais fácil de entender para o programador iniciante. Além do mais, diferentemente do Java, ele fornece acesso a *sockets* brutos, permitindo que os alunos construam maior variedade de aplicações de rede. Laboratórios de programação de *sockets* baseados em Java foram substituídos por laboratórios Python correspondentes, e foi acrescentado um novo laboratório ICMP Ping baseado em Python. Como sempre, quando um material é retirado — como aquele sobre programação de *sockets* baseada em Java — ele permanece disponível no site de apoio do livro (ver texto mais adiante).
- No Capítulo 3, a apresentação de um dos protocolos de transferência de dados confiável foi simplificada e uma nova nota em destaque sobre divisão do TCP, normalmente usada para otimizar o desempenho de serviços de nuvem, foi acrescentada.

- No Capítulo 4, a seção sobre arquiteturas de roteador foi significativamente atualizada, refletindo desenvolvimentos e práticas recentes nessa área. Incluímos diversas novas notas em destaque, envolvendo DNS, BGP e OSPF.
- O Capítulo 5 foi reorganizado e simplificado, considerando a onipresença da Ethernet comutada em redes locais e o consequente aumento do uso da Ethernet em cenários ponto a ponto. Além disso, acrescentamos uma seção sobre redes de centro de dados.
- O Capítulo 6 foi atualizado para refletir os avanços recentes nas redes sem fio, em particular redes de dados por celular e serviços e arquiteturas 4G.
- O Capítulo 7, que enfoca as redes multimídia, passou por uma grande revisão. Ele agora contém uma discussão profunda do vídeo de fluxo contínuo (*streaming*), incluindo fluxo contínuo adaptativo, e uma discussão nova e modernizada de CDNs. Uma seção recém-incluída descreve os sistemas de vídeo de fluxo contínuo Netflix, YouTube e Kankan. O material que foi removido para dar espaço para esses novos tópicos ainda está disponível no site de apoio.
- O Capítulo 8 agora contém uma discussão expandida sobre autenticação do ponto final.
- Acrescentamos um novo material significativo, envolvendo problemas de fim de capítulo. Assim como em todas as edições anteriores, trabalhos de casa foram revisados, acrescentados e removidos.

PÚBLICO-ALVO

Este livro destina-se a um estudo inicial de redes de computadores. Pode ser usado em cursos de ciência da computação e de engenharia elétrica. Em termos de linguagens de programação, ele requer que os estudantes tenham alguma experiência com as linguagens C, C++, Java ou Python (mesmo assim, apenas em alguns lugares). Embora seja mais minucioso e analítico do que muitos outros de introdução às redes de computadores, raramente utiliza conceitos matemáticos que não sejam ensinados no ensino médio. Fizemos um esforço deliberado para evitar o uso de quaisquer conceitos avançados de cálculo, probabilidade ou processos estocásticos (embora tenhamos incluído alguns problemas para alunos com tal conhecimento avançado). Por conseguinte, o livro é apropriado para cursos de graduação e para o primeiro ano dos cursos de pós-graduação. É também muito útil para os profissionais do setor de telecomunicações.

O QUE HÁ DE SINGULAR NESTE LIVRO?

O assunto rede de computadores é bastante vasto e complexo, envolvendo muitos conceitos, protocolos e tecnologias que se entrelaçam inextricavelmente. Para dar conta desse escopo e complexidade, muitos livros sobre redes são, em geral, organizados de acordo com as "camadas" de uma arquitetura de rede. Com a organização em camadas, os estudantes podem vislumbrar a complexidade das redes de computadores — eles aprendem os conceitos e os protocolos distintos de uma parte da arquitetura e, ao mesmo tempo, visualizam o grande quadro da interconexão entre as camadas. Do ponto de vista pedagógico, nossa experiência confirma que essa abordagem em camadas é, de fato, muito boa. Entretanto, achamos que a abordagem tradicional, a *bottom-up* — da camada física para a camada de aplicação —, não é a melhor para um curso moderno de redes de computadores.

UMA ABORDAGEM TOP-DOWN

Na primeira edição, propusemos uma inovação adotando uma visão *top-down* — isto é, começando na camada de aplicação e descendo até a camada física. O retorno que recebemos de professores e alunos confirmou que essa abordagem tem muitas vantagens e realmente funciona bem do ponto de vista pedagógico. Primeiro, o livro dá ênfase à camada de aplicação, que tem sido a área de "grande crescimento" das redes de computadores. De fato, muitas das recentes revoluções nesse ramo — incluindo a Web, o compartilhamento de arquivos P2P e o fluxo contínuo de mídia — tiveram lugar nessa camada. A abordagem de ênfase inicial à camada de aplicação é diferente das seguidas por muitos outros livros, que têm apenas pouco material sobre aplicações de redes, seus requisitos, paradigmas da camada de aplicação (por exemplo, cliente-servidor e P2P) e interfaces de programação

de aplicação. Segundo, nossa experiência como professores (e a de muitos outros que utilizaram este livro) confirma que ensinar aplicações de rede logo no início do curso é uma poderosa ferramenta motivadora. Os estudantes ficam mais entusiasmados ao aprender como funcionam as aplicações de rede — aplicações como o e-mail e a Web, que a maioria deles usa diariamente. Entendendo as aplicações, o estudante pode entender os serviços de rede necessários ao suporte de tais aplicações. Pode também, por sua vez, examinar as várias maneiras como esses serviços são fornecidos e executados nas camadas mais baixas. Assim, a discussão das aplicações logo no início fornece a motivação necessária para os demais assuntos do livro.

Terceiro, a abordagem *top-down* habilita o professor a apresentar o desenvolvimento das aplicações de rede no estágio inicial. Os estudantes não só veem como funcionam aplicações e protocolos populares, como também aprendem que é fácil criar suas próprias aplicações e protocolos de aplicação de rede. Com a abordagem *top-down*, eles entram imediatamente em contato com as noções de programação de *sockets*, modelos de serviços e protocolos — conceitos importantes que reaparecem em todas as camadas subsequentes. Ao apresentar exemplos de programação de *sockets* em Python, destacamos as ideias centrais sem confundir os estudantes com códigos complexos. Estudantes de engenharia elétrica e ciência da computação talvez não tenham dificuldades para entender o código Python.

UM FOCO NA INTERNET

Continuamos a colocar a Internet em foco nesta edição do livro. Na verdade, como ela está presente em toda parte, achamos que qualquer livro sobre redes deveria ter um foco significativo na Internet. Continuamos a utilizar a arquitetura e os protocolos da Internet como veículo primordial para estudar conceitos fundamentais de redes de computadores. É claro que também incluímos conceitos e protocolos de outras arquiteturas de rede. Mas os holofotes estão claramente dirigidos à Internet, fato refletido na organização do livro, que gira em torno da arquitetura de cinco camadas: aplicação, transporte, rede, enlace e física.

Outro benefício de colocá-la sob os holofotes é que a maioria dos estudantes de ciência da computação e de engenharia elétrica está ávida por conhecer a Internet e seus protocolos. Eles sabem que a Internet é uma tecnologia revolucionária e inovadora e podem constatar que ela está provocando uma profunda transformação em nosso mundo. Dada sua enorme relevância, os estudantes estão naturalmente curiosos em saber o que há por trás dela. Assim, fica fácil para um professor manter seus alunos interessados nos princípios básicos, usando a Internet como guia.

ENSINANDO PRINCÍPIOS DE REDE

Duas das características exclusivas deste livro — sua abordagem *top-down* e seu foco na Internet — aparecem no título e subtítulo. Se pudéssemos, teríamos acrescentado uma *terceira* palavra — *princípios*. O campo das redes agora está maduro o suficiente para que uma quantidade de assuntos de importância fundamental possa ser identificada. Por exemplo, na camada de transporte, entre os temas importantes estão a comunicação confiável por uma camada de rede não confiável, o estabelecimento/encerramento de conexões e mútua apresentação (*handshaking*), o controle de congestionamento e de fluxo e a multiplexação. Na camada de rede, dois assuntos muito importantes são: como determinar "bons" caminhos entre dois roteadores e como interconectar um grande número de redes heterogêneas. Na camada de enlace, um problema fundamental é como compartilhar um canal de acesso múltiplo. Na segurança de rede, técnicas para prover sigilo, autenticação e integridade de mensagens são baseadas em fundamentos da criptografia. Este livro identifica as questões fundamentais de redes e apresenta abordagens para enfrentar tais questões. Aprendendo esses princípios, o estudante adquire conhecimento de "longa validade" — muito tempo após os padrões e protocolos de rede de hoje tornarem-se obsoletos, os princípios que ele incorpora continuarão importantes e relevantes. Acreditamos que o uso da Internet para apresentar o assunto aos estudantes e a ênfase dada à abordagem das questões e das soluções permitirão que os alunos entendam rapidamente qualquer tecnologia de rede.

CARACTERÍSTICAS PEDAGÓGICAS

Há quase 20 anos damos aulas de redes de computadores. Adicionamos a este livro uma experiência agregada de mais de 50 anos de ensino para milhares de estudantes. Nesse período, também participamos ativamente na área de pesquisas sobre redes de computadores. (De fato, Jim e Keith se conheceram quando faziam mestrado, frequentando um curso sobre redes de computadores ministrado por Mischa Schwartz, em 1979, na Universidade de Colúmbia.) Achamos que isso nos dá uma boa perspectiva do que foi a rede e de qual será, provavelmente, seu futuro. Não obstante, resistimos às tentações de dar ao material deste livro um viés que favorecesse nossos projetos de pesquisa prediletos. Se você estiver interessado em nossas pesquisas, consulte nosso site pessoal. Este livro é sobre redes de computadores modernas — é sobre protocolos e tecnologias contemporâneas, bem como sobre os princípios subjacentes a esses protocolos e tecnologias. Também achamos que aprender (e ensinar!) redes pode ser divertido. Esperamos que algum senso de humor e a utilização de analogias e exemplos do mundo real que aparecem aqui tornem o material ainda mais divertido.

DEPENDÊNCIAS DE CAPÍTULO

O primeiro capítulo apresenta um apanhado geral sobre redes de computadores. Com a introdução de muitos conceitos e terminologias fundamentais, ele monta o cenário para o restante do livro. Todos os outros capítulos dependem diretamente desse primeiro. Recomendamos que os professores, após o terem completado, percorram em sequência os Capítulos 2 ao 5, seguindo nossa filosofia *top-down*. Cada um dos cinco primeiros capítulos utiliza material dos anteriores. Após tê-los completado, o professor terá bastante flexibilidade. Não há interdependência entre os quatro últimos capítulos, de modo que eles podem ser ensinados em qualquer ordem. Porém, cada um dos quatro últimos capítulos depende do material nos cinco primeiros. Muitos professores a princípio ensinam os cinco primeiros capítulos e depois ensinam um dos quatro últimos para "arrematar".

AGRADECIMENTOS

Desde o início deste projeto, em 1996, muitas pessoas nos deram inestimável auxílio e influenciaram nossas ideias sobre como melhor organizar e ministrar um curso sobre redes. Nosso MUITO OBRIGADO a todos os que nas ajudaram desde os primeiros rascunhos deste livro, até a quinta edição. Também somos *muito* gratos sobretudo às muitas centenas de leitores de todo o mundo — estudantes, acadêmicos e profissionais — que nos enviaram sugestões e comentários sobre edições anteriores e sugestões para edições futuras. Nossos agradecimentos especiais para:

Al Aho (Universidade de Columbia)
Hisham Al-Mubaid (Universidade de Houston-Clear Lake)
Pratima Akkunoor (Universidade do Estado do Arizona)
Paul Amer (Universidade de Delaware)
Shamiul Azom (Universidade do Estado do Arizona)
Lichun Bao (Universidade da Califórnia, Irvine)
Paul Barford (Universidade do Wisconsin)
Bobby Bhattacharjee (Universidade de Maryland)
Steven Bellovin (Universidade de Columbia)
Pravin Bhagwat (Wibhu)
Supratik Bhattacharyya (anteriormente na Sprint)
Ernst Biersack (Eurécom Institute)
Shahid Bokhari (Universidade de Engenharia & Tecnologia, Lahore)
Jean Bolot (Technicolor Research)

Daniel Brushteyn (ex-aluno da Universidade da Pensilvânia)
Ken Calvert (Universidade do Kentucky)
Evandro Cantu (Universidade Federal de Santa Catarina)
Jeff Case (SNMP Research International)
Jeff Chaltas (Sprint)
Vinton Cerf (Google)
Byung Kyu Choi (Universidade Tecnológica de Michigan)
Bram Cohen (BitTorrent, Inc.)
Constantine Coutras (Universidade Pace)
John Daigle (Universidade do Mississippi)
Edmundo A. de Souza e Silva (Universidade Federal do Rio de Janeiro)
Philippe Decuetos (Eurécom Institute)
Christophe Diot (Technicolor Research)
Prithula Dhunghel (Akamai)
Deborah Estrin (Universidade da Califórnia, Los Angeles)
Michalis Faloutsos (Universidade da Califórnia, Riverside)
Wu-chi Feng (Oregon Graduate Institute)
Sally Floyd (ICIR, Universidade da Califórnia, Berkeley)
Paul Francis (Max Planck Institute)
Lixin Gao (Universidade de Massachusetts)
JJ Garcia-Luna-Aceves (Universidade da Califórnia, Santa Cruz)
Mario Gerla (Universidade da Califórnia, Los Angeles)
David Goodman (NYU-Poly)
Yang Guo (Alcatel/Lucent Bell Labs)
Tim Griffin (Universidade de Cambridge)
Max Hailperin (Gustavus Adolphus College)
Bruce Harvey (Universidade da Flórida A&M)
Carl Hauser (Universidade do Estado de Washington)
Rachelle Heller (Universidade George Washington)
Phillipp Hoschka (INRIA/W3C)
Wen Hsin (Universidade Park)
Albert Huang (ex-aluno da Universidade da Pensilvânia)
Cheng Huang (Microsoft Research)
Esther A. Hughes (Universidade Virginia Commonwealth)
Van Jacobson (Xerox PARC)
Pinak Jain (ex-aluno da NYU-Poly)
Jobin James (Universidade da Califórnia, Riverside)
Sugih Jamin (Universidade de Michigan)
Shivkumar Kalyanaraman (IBM Research, Índia)
Jussi Kangasharju (Universidade de Helsinki)
Sneha Kasera (Universidade de Utah)
Parviz Kermani (anteriormente da IBM Research)
Hyojin Kim (ex-aluno da Universidade da Pensilvânia)

Leonard Kleinrock (Universidade da Califórnia, Los Angeles)
David Kotz (Dartmouth College)
Beshan Kulapala (Universidade do Estado do Arizona)
Rakesh Kumar (Bloomberg)
Miguel A. Labrador (Universidade South Florida)
Simon Lam (Universidade do Texas)
Steve Lai (Universidade do Estado de Ohio)
Tom LaPorta (Universidade do Estado de Pensilvânia)
Tim-Berners Lee (World Wide Web Consortium)
Arnaud Legout (INRIA)
Lee Leitner (Universidade Drexel)
Brian Levine (Universidade de Massachusetts)
Chunchun Li (ex-aluno da NYU-Poly)
Yong Liu (NYU-Poly)
William Liang (ex-aluno da Universidade da Pensilvânia)
Willis Marti (Universidade Texas A&M)
Nick McKeown (Universidade Stanford)
Josh McKinzie (Universidade Park)
Deep Medhi (Universidade do Missouri, Kansas City)
Bob Metcalfe (International Data Group)
Sue Moon (KAIST)
Jenni Moyer (Comcast)
Erich Nahum (IBM Research)
Christos Papadopoulos (Universidade do Estado do Colorado)
Craig Partridge (BBN Technologies)
Radia Perlman (Intel)
Jitendra Padhye (Microsoft Research)
Vern Paxson (Universidade da Califórnia, Berkeley)
Kevin Phillips (Sprint)
George Polyzos (Universidade de Economia e Negócios de Atenas)
Sriram Rajagopalan (Universidade do Estado do Arizona)
Ramachandran Ramjee (Microsoft Research)
Ken Reek (Rochester Institute of Technology)
Martin Reisslein (Universidade do Estado do Arizona)
Jennifer Rexford (Universidade de Princeton)
Leon Reznik (Rochester Institute of Technology)
Pablo Rodrigez (Telefonica)
Sumit Roy (Universidade de Washington)
Avi Rubin (Universidade Johns Hopkins)
Dan Rubenstein (Universidade de Columbia)
Douglas Salane (John Jay College)
Despina Saparilla (Cisco Systems)
John Schanz (Comcast)

Henning Schulzrinne (Universidade de Columbia)
Mischa Schwartz (Universidade de Columbia)
Ardash Sethi (Universidade de Delaware)
Harish Sethu (Universidade Drexel)
K. Sam Shanmugan (Universidade do Kansas)
Prashant Shenoy (Universidade de Massachusetts)
Clay Shields (Universidade Georgetown)
Subin Shrestra (Universidade da Pensilvânia)
Bojie Shu (ex-aluno da NYU-Poly)
Mihail L. Sichitiu (Universidade do Estado de NC)
Peter Steenkiste (Universidade Carnegie Mellon)
Tatsuya Suda (Universidade da Califórnia, Irvine)
Kin Sun Tam (Universidade do Estado de Nova York, Albany)
Don Towsley (Universidade de Massachusetts)
David Turner (Universidade do Estado da Califórnia, San Bernardino)
Nitin Vaidya (Universidade de Illinois)
Michele Weigle (Universidade Clemson)
David Wetherall (Universidade de Washington)
Ira Winston (Universidade da Pensilvânia)
Di Wu (Universidade Sun Yat-sen)
Shirley Wynn (NYU-Poly)
Raj Yavatkar (Intel)
Yechiam Yemini (Universidade de Columbia)
Ming Yu (Universidade do Estado de Nova York, Binghamton)
Ellen Zegura (Instituto de Tecnologia da Geórgia)
Honggang Zhang (Universidade Suffolk)
Hui Zhang (Universidade Carnegie Mellon)
Lixia Zhang (Universidade da Califórnia, Los Angeles)
Meng Zhang (ex-aluno da NYU-Poly)
Shuchun Zhang (ex-aluno da Universidade da Pensilvânia)
Xiaodong Zhang (Universidade do Estado de Ohio)
ZhiLi Zhang (Universidade de Minnesota)
Phil Zimmermann (consultor independente)
Cliff C. Zou (Universidade Central Florida)

Queremos agradecer, também, a toda a equipe da Addison-Wesley — em particular, a Michael Hirsch, Marilyn Lloyd e Emma Snider — que fez um trabalho realmente notável nesta sexta edição (e que teve de suportar dois autores muito complicados e quase sempre atrasados). Agradecemos aos artistas gráficos Janet Theurer e Patrice Rossi Calkin, pelo trabalho que executaram nas belas figuras deste livro, e a Andrea Stefanowicz e sua equipe na PreMediaGlobal, pelo maravilhoso trabalho de produção desta edição. Por fim, um agradecimento muito especial a Michael Hirsch, nosso editor na Addison-Wesley, e a Susan Hartman, nossa antiga editora. Este livro não seria o que é (e talvez nem tivesse existido) sem a administração cordial de ambos, constante incentivo, paciência quase infinita, bom humor e perseverança.

Agradecimentos — Edição brasileira

Agradecemos a todos os profissionais envolvidos na produção desta edição no Brasil, em especial ao Prof. Dr. Wagner Luiz Zucchi (Escola Politécnica da USP, Instituto de Pesquisas Tecnológicas — IPT — e Universidade Nove de Julho), por sua dedicação e empenho na revisão técnica do conteúdo.

Materiais adicionais

A Sala Virtual (<sv.pearson.com.br>) oferece recursos adicionais que auxiliarão professores e alunos na exposição das aulas e no processo de aprendizagem.

Para o professor:

- Apresentações em PowerPoint
- Manual de soluções (em inglês)

Para o aluno:

- Exercícios autocorrigíveis (10 por capítulo)
- Exercícios interativos
- Material de aprendizagem interativo (applets e códigos-fonte)
- Tarefas extras de programação – Python e Java (em inglês)
- Wireshark labs (em inglês)
- Exemplos para implementação de laboratórios (em inglês)

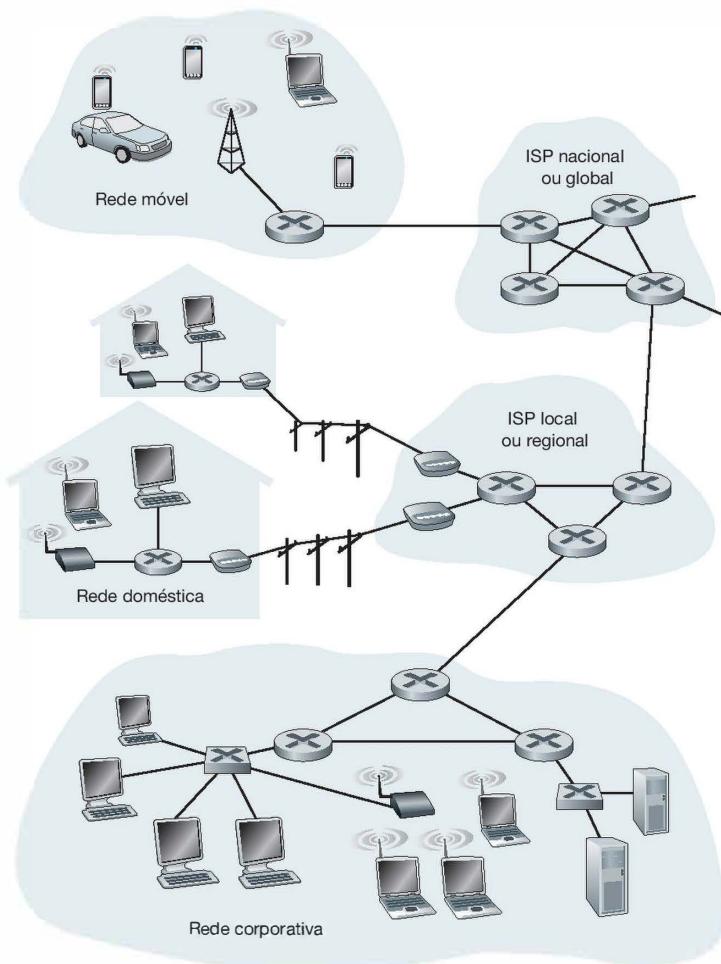
SUMÁRIO

C A P Í T U L O

1 2 3 4 5 6 7 8 9

REDES DE COMPUTADORES E A INTERNET

A Internet de hoje é provavelmente o maior sistema de engenharia já criado pela humanidade, com centenas de milhões de computadores conectados, enlaces de comunicação e comutadores; bilhões de usuários que se conectam por meio de laptops, tablets e smartphones; e com uma série de dispositivos como sensores, webcams, console para jogos, quadros de imagens, e até mesmo máquinas de lavar sendo conectadas. Dado que a Internet é tão ampla e possui inúmeros componentes e utilidades, há a possibilidade de compreender como ela funciona? Existem princípios de orientação e estrutura que forneçam um fundamento para a compreensão de um sistema surpreendentemente complexo e abrangente? Se a resposta for sim, é possível que, nos dias de hoje, seja interessante *e* divertido aprender sobre rede de computadores? Felizmente, as respostas para todas essas perguntas é um retumbante SIM! Na verdade, nosso objetivo neste livro é fornecer uma introdução moderna ao campo dinâmico das redes de computadores, apresentando os princípios e o entendimento prático necessários para utilizar não apenas as redes de hoje, como também as de amanhã.

O primeiro capítulo apresenta um panorama de redes de computadores e da Internet. Nosso objetivo é pintar um quadro amplo e estabelecer um contexto para o resto deste livro, para ver a floresta por entre as árvores. Cobriremos um terreno bastante extenso neste capítulo de introdução e discutiremos várias peças de uma rede de computadores, sem perder de vista o quadro geral.

O panorama geral de redes de computadores que apresentaremos neste capítulo será estruturado como segue. Após apresentarmos brevemente a terminologia e os conceitos fundamentais, examinaremos primeiro os componentes básicos de hardware e software que compõem uma rede. Partiremos da periferia da rede e examinaremos os sistemas finais e aplicações de rede executados nela. Consideraremos os serviços de transporte fornecidos a essas aplicações. Em seguida exploraremos o núcleo de uma rede de computadores examinando os enlaces e comutadores que transportam dados, bem como as redes de acesso e meios físicos que conectam sistemas finais ao núcleo da rede. Aprenderemos que a Internet é uma rede de redes e observaremos como essas redes se conectam umas com as outras.

Após concluirmos essa revisão sobre a periferia e o núcleo de uma rede de computadores, adotaremos uma visão mais ampla e mais abstrata na segunda metade deste capítulo. Examinaremos atraso, perda e vazão de dados em uma rede de computadores e forneceremos modelos quantitativos simples para a vazão e o atraso fim a fim: modelos que levam em conta atrasos de transmissão, propagação e enfileiramento. Depois apresentaremos alguns princípios fundamentais de arquitetura em redes de computadores, a saber: protocolos em camadas e modelos de serviço. Aprenderemos, também, que as redes de computadores são vulneráveis a diferentes tipos de ameaças; analisaremos algumas dessas ameaças e como a rede pode se tornar mais segura. Por fim, encerraremos este capítulo com um breve histórico da computação em rede.

1.1 O QUE É A INTERNET?

Neste livro, usamos a Internet pública, uma rede de computadores específica, como o veículo principal para discutir as redes de computadores e seus protocolos. Mas o que *é* a Internet? Há diversas maneiras de responder a essa questão. Primeiro, podemos descrever detalhadamente os aspectos principais da Internet, ou seja, os componentes de software e hardware básicos que a formam. Segundo, podemos descrever a Internet em termos de uma infraestrutura de redes que fornece serviços para aplicações distribuídas. Iniciaremos com a descrição dos componentes, utilizando a Figura 1.1 como ilustração para a nossa discussão.

FIGURA 1.1 ALGUNS COMPONENTES DA INTERNET

1.1.1 Uma descrição dos componentes da rede

A Internet é uma rede de computadores que interconecta centenas de milhões de dispositivos de computação ao redor do mundo. Há pouco tempo, esses dispositivos eram basicamente PCs de mesa, estações de trabalho Linux, e os assim chamados servidores que armazenam e transmitem informações, como páginas da Web e mensagens de e-mail. No entanto, cada vez mais sistemas finais modernos da Internet, como TVs, laptops, consoles para jogos, telefones celulares, *webcams*, automóveis, dispositivos de sensoriamento ambiental, quadros de imagens, e sistemas internos elétricos e de segurança, estão sendo conectados à rede. Na verdade, o termo *rede de computadores* está começando a soar um tanto desatualizado, dados os muitos equipamentos não tradicionais que estão sendo ligados à Internet. No jargão da rede, todos esses equipamentos são denominados **hospedeiros** ou **sistemas finais**. Em julho de 2011, havia cerca de 850 milhões de sistemas finais ligados à Internet [ISC, 2012], sem contar os smartphones, laptops e outros dispositivos que são conectados à rede de maneira intermitente. No todo, estima-se que haja 2 bilhões de usuários na Internet [ITU, 2011].

Sistemas finais são conectados entre si por **enlaces (*links*) de comunicação** e **comutadores (*switches*) de pacotes**. Na Seção 1.2, veremos que há muitos tipos de enlaces de comunicação, que são constituídos de diferentes tipos de meios físicos, entre eles cabos coaxiais, fios de cobre, fibras óticas e ondas de rádio. Enlaces diferentes podem transmitir dados em taxas diferentes, sendo a **taxa de transmissão** de um enlace medida em bits por segundo. Quando um sistema final possui dados para enviar a outro sistema final, o sistema emissor segmenta esses dados e adiciona bytes de cabeçalho a cada segmento. Os pacotes de informações resultantes, conhecidos como **pacotes** no jargão de rede de computadores, são enviados através da rede ao sistema final de destino, onde são remontados para os dados originais.

Um comutador de pacotes encaminha o pacote que está chegando em um de seus enlaces de comunicação de entrada para um de seus enlaces de comunicação de saída. Há comutadores de pacotes de todos os tipos e formas, mas os dois mais proeminentes na Internet de hoje são **roteadores** e **comutadores de camada de enlace**. Esses dois tipos de comutadores encaminham pacotes a seus destinos finais. Os comutadores de camada de enlace geralmente são utilizados em redes de acesso, enquanto os roteadores são utilizados principalmente no núcleo da rede. A sequência de enlaces de comunicação e comutadores de pacotes que um pacote percorre desde o sistema final remetente até o sistema final receptor é conhecida como **rota** ou **caminho** através da rede. É difícil de estimar a exata quantidade de tráfego na Internet, mas a Cisco [Cisco VNI, 2011] estima que o tráfego global da Internet esteve perto do 40 exabytes por mês em 2012.

As redes comutadas por pacotes (que transportam pacotes) são, de muitas maneiras, semelhantes às redes de transporte de rodovias, estradas e cruzamentos (que transportam veículos). Considere, por exemplo, uma fábrica que precise transportar uma quantidade de carga muito grande a algum depósito localizado a milhares de quilômetros. Na fábrica, a carga é dividida e carregada em uma frota de caminhões. Cada caminhão viaja, de modo independente, pela rede de rodovias, estradas e cruzamentos ao depósito de destino. No depósito, a carga é descarregada e agrupada com o resto da carga pertencente à mesma remessa. Deste modo, os pacotes se assemelham aos caminhões, os enlaces de comunicação representam rodovias e estradas, os comutadores de pacote seriam os cruzamentos e cada sistema final se assemelha aos depósitos. Assim como o caminhão faz o percurso pela rede de transporte, o pacote utiliza uma rede de computadores.

Sistemas finais acessam a Internet por meio de **Provedores de Serviços de Internet** (*Internet Service Providers* — ISPs), entre eles ISPs residenciais como empresas de TV a cabo ou empresas de telefonia; corporativos, de universidades e ISPs que fornecem acesso sem fio em aeroportos, hotéis, cafés e outros locais públicos. Cada ISP é uma rede de comutadores de pacotes e enlaces de comunicação. ISPs oferecem aos sistemas finais uma variedade de tipos de acesso à rede, incluindo acesso residencial de banda larga como modem a cabo ou DSL (linha digital de assinante), acesso por LAN de alta velocidade, acesso sem fio e acesso por modem discado de 56 kbits/s. ISPs também fornecem acesso a provedores de conteúdo, conectando sites diretamente à Internet. Esta se interessa pela conexão entre os sistemas finais, portanto os ISPs que fornecem acesso a esses sistemas também devem se interconectar. Esses ISPs de nível mais baixo são interconectados por meio de ISPs de nível mais alto, nacionais e internacionais, como Level 3 Communications, AT&T, Sprint e NTT. Um ISP de nível mais alto consiste em roteadores de alta velocidade

interconectados com enlaces de fibra ótica de alta velocidade. Cada rede ISP, seja de nível mais alto ou mais baixo, é gerenciada de forma independente, executa o protocolo IP (ver adiante) e obedece a certas convenções de nomeação e endereço. Examinaremos ISPs e sua interconexão mais em detalhes na Seção 1.3.

Os sistemas finais, os comutadores de pacotes e outras peças da Internet executam **protocolos** que controlam o envio e o recebimento de informações. O **TCP** (**Transmission Control Protocol** — Protocolo de Controle de Transmissão) e o **IP** (**Internet Protocol** — Protocolo da Internet) são dois dos mais importantes da Internet. O protocolo IP especifica o formato dos pacotes que são enviados e recebidos entre roteadores e sistemas finais. Os principais protocolos da Internet são conhecidos como **TCP/IP**. Começaremos a examinar protocolos neste capítulo de introdução. Mas isso é só um começo — grande parte deste livro trata de protocolos de redes de computadores!

Dada a importância de protocolos para a Internet, é adequado que todos concordem sobre o que cada um deles faz, de modo que as pessoas possam criar sistemas e produtos que operem entre si. É aqui que os padrões entram em ação. **Padrões da Internet** são desenvolvidos pela IETF (Internet Engineering Task Force — Força de Trabalho de Engenharia da Internet) [IETF, 2012]. Os documentos padronizados da IETF são denominados **RFCs** (**Request For Comments** — pedido de comentários). Os RFCs começaram como solicitações gerais de comentários (daí o nome) para resolver problemas de arquitetura que a precursora da Internet enfrentava [Allman, 2011]. Os RFCs costumam ser bastante técnicos e detalhados. Definem protocolos como TCP, IP, HTTP (para a Web) e SMTP (para e-mail). Hoje, existem mais de 6.000 RFCs. Outros órgãos também especificam padrões para componentes de rede, principalmente para enlaces. O IEEE 802 LAN/MAN Standards Committee [IEEE 802, 2009], por exemplo, especifica os padrões Ethernet e Wi-Fi sem fio.

1.1.2 Uma descrição do serviço

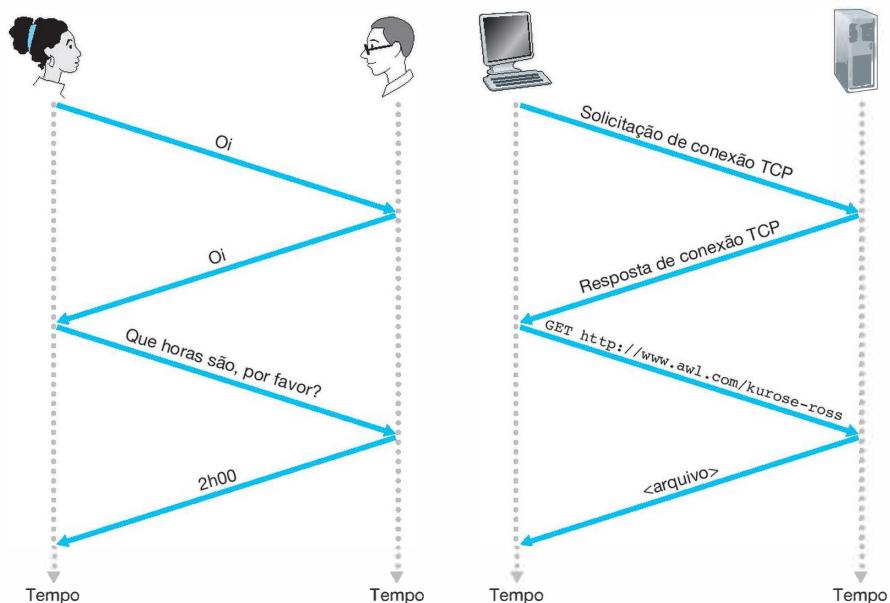
A discussão anterior identificou muitos dos componentes que compõem a Internet. Mas também podemos descrevê-la partindo de um ângulo completamente diferente — ou seja, como *uma infraestrutura que provê serviços a aplicações*. Tais aplicações incluem correio eletrônico, navegação na Web, redes sociais, mensagem instantânea, Voz sobre IP (VoIP), vídeo em tempo real, jogos distribuídos, compartilhamento de arquivos *peer-to-peer* (P2P), televisão pela Internet, login remoto e muito mais. Essas aplicações são conhecidas como **aplicações distribuídas**, uma vez que envolvem diversos sistemas finais que trocam informações mutuamente. De forma significativa, as aplicações da Internet são executadas em sistemas finais — e não em comutadores de pacote no núcleo da rede. Embora os comutadores de pacotes facilitem a troca de dados entre os sistemas finais, eles não estão relacionados com a aplicação, que é a origem ou o destino dos dados.

Vamos explorar um pouco mais o significado de uma infraestrutura que fornece serviços a aplicações. Nesse sentido, suponha que você tenha uma grande ideia para uma aplicação distribuída para a Internet, uma que possa beneficiar bastante a humanidade ou que simplesmente o enriqueça e o torne famoso. Como transformar essa ideia em uma aplicação real da Internet? Como as aplicações são executadas em sistemas finais, você precisará criar programas que sejam executados em sistemas finais. Você poderia, por exemplo, criar seus programas em Java, C ou Python. Agora, já que você está desenvolvendo uma aplicação distribuída para a Internet, os programas executados em diferentes sistemas finais precisarão enviar dados uns aos outros. E, aqui, chegamos ao assunto principal — o que leva ao modo alternativo de descrever a Internet como uma plataforma para aplicações. De que modo um programa, executado em um sistema final, orienta a Internet a enviar dados a outro programa executado em outro sistema final?

Os sistemas finais ligados à Internet oferecem uma **Interface de Programação de Aplicação** (API) que especifica como o programa que é executado no sistema final solicita à infraestrutura da Internet que envie dados a um programa de destino específico, executado em outro sistema final. Essa API da Internet é um conjunto de regras que o software emissor deve cumprir para que a Internet seja capaz de enviar os dados ao programa de destino. Discutiremos a API da Internet mais detalhadamente no Capítulo 2. Agora, vamos traçar uma simples comparação, que será utilizada com frequência neste livro. Suponha que Alice queria enviar uma carta para Bob utilizando o serviço postal. Alice, é claro, não pode apenas escrever a carta (os dados) e atirá-la pela janela.

Em vez disso, o serviço postal necessita que ela coloque a carta em um envelope; escreva o nome completo de Bob, endereço e CEP no centro do envelope; feche; coloque um selo no canto superior direito; e, por fim, leve o envelope a uma agência de correio oficial. Dessa maneira, o serviço postal possui sua própria "API de serviço postal", ou conjunto de regras, que Alice deve cumprir para que sua carta seja entregue a Bob. De um modo semelhante, a Internet possui uma API que o software emissor de dados deve seguir para que a Internet envie os dados para o software receptor.

O serviço postal, claro, fornece mais de um serviço a seus clientes: entrega expressa, aviso de recebimento, carta simples e muito mais. De modo semelhante, a Internet provê diversos serviços a suas aplicações. Ao desenvolver uma aplicação para a Internet, você também deve escolher um dos serviços que a rede oferece. Uma descrição dos serviços será apresentada no Capítulo 2.

Acabamos de apresentar duas descrições da Internet: uma delas diz respeito a seus componentes de hardware e software, e a outra, aos serviços que ela oferece a aplicações distribuídas. Mas talvez você ainda esteja confuso sobre o que é a Internet. O que é comutação de pacotes e TCP/IP? O que são roteadores? Que tipos de enlaces de comunicação estão presentes na Internet? O que é uma aplicação distribuída? Como uma torradeira ou um sensor de variações meteorológicas podem ser ligados à Internet? Se você está um pouco assustado com tudo isso agora, não se preocupe — a finalidade deste livro é lhe apresentar os mecanismos da Internet e também os princípios que determinam como e por que ela funciona. Explicaremos esses termos e questões importantes nas seções e nos capítulos subsequentes.

1.1.3 O que é um protocolo?

Agora que já entendemos um pouco o que é a Internet, vamos considerar outra palavra fundamental usada em redes de computadores: *protocolo*. O que é um protocolo? O que um protocolo faz?

Uma analogia humana

Talvez seja mais fácil entender a ideia de um protocolo de rede de computadores considerando primeiro algumas analogias humanas, já que executamos protocolos o tempo todo. Considere o que você faz quando quer perguntar as horas a alguém. Um diálogo comum é ilustrado na Figura 1.2. O protocolo humano (ou as boas maneiras, ao menos) dita que, ao iniciarmos uma comunicação com outra pessoa, primeiro a cumprimentemos (o primeiro "oi" da Figura 1.2). A resposta comum para um "oi" é um outro "oi". Implicitamente, tomamos a resposta cordial "oi" como uma indicação de que podemos prosseguir e perguntar as horas. Uma reação diferente ao "oi" inicial (tal como "Não me perturbe!", "*I don't speak Portuguese*!" ou alguma resposta atravessada) poderia indicar falta de vontade ou incapacidade de comunicação. Nesse caso, o protocolo humano seria não perguntar que horas são. Às vezes, não recebemos nenhuma resposta para uma pergunta, caso em que em geral desistimos de perguntar as horas à pessoa. Note que, no nosso protocolo humano, *há mensagens específicas que enviamos e ações específicas que realizamos em reação às respostas recebidas ou a outros eventos* (como nenhuma resposta após certo tempo). É claro que mensagens transmitidas e recebidas e ações realizadas quando essas mensagens são enviadas ou recebidas ou quando ocorrem outros eventos desempenham um papel central em um protocolo humano. Se as pessoas executarem protocolos diferentes (por exemplo, se uma pessoa tem boas maneiras, mas a outra não; se uma delas entende o conceito de horas, mas a outra não), os protocolos não interagem e nenhum trabalho útil pode ser realizado. O mesmo é válido para redes — é preciso que duas (ou mais) entidades comunicantes executem o mesmo protocolo para que uma tarefa seja realizada.

Vamos considerar uma segunda analogia humana. Suponha que você esteja assistindo a uma aula (sobre redes de computadores, por exemplo). O professor está falando monotonamente sobre protocolos e você está confuso. Ele para e pergunta: "Alguma dúvida?" (uma mensagem que é transmitida a todos os alunos e recebida por todos os que não estão dormindo). Você levanta a mão (transmitindo uma mensagem implícita ao professor). O professor percebe e, com um sorriso, diz "Sim..." (uma mensagem transmitida, incentivando-o a fazer sua

FIGURA 1.2 UM PROTOCOLO HUMANO E UM PROTOCOLO DE REDE DE COMPUTADORES

pergunta — professores *adoram* perguntas) e você então faz a sua (isto é, transmite sua mensagem ao professor). Ele ouve (recebe sua mensagem) e responde (transmite uma resposta a você). Mais uma vez, percebemos que a transmissão e a recepção de mensagens e um conjunto de ações convencionais, realizadas quando as mensagens são enviadas e recebidas, estão no centro desse protocolo de pergunta e resposta.

Protocolos de rede

Um protocolo de rede é semelhante a um protocolo humano; a única diferença é que as entidades que trocam mensagens e realizam ações são componentes de hardware ou software de algum dispositivo (por exemplo, computador, smartphone, tablet, roteador ou outro equipamento habilitado para rede). Todas as atividades na Internet que envolvem duas ou mais entidades remotas comunicantes são governadas por um protocolo. Por exemplo, protocolos executados no hardware de dois computadores conectados fisicamente controlam o fluxo de bits no "cabo" entre as duas placas de interface de rede; protocolos de controle de congestionamento em sistemas finais controlam a taxa com que os pacotes são transmitidos entre a origem e o destino; protocolos em roteadores determinam o caminho de um pacote da origem ao destino. Eles estão em execução por toda a Internet e, em consequência, grande parte deste livro trata de protocolos de rede de computadores.

Como exemplo de um protocolo de rede de computadores com o qual você provavelmente está familiarizado, considere o que acontece quando fazemos uma requisição a um servidor Web, isto é, quando digitamos o URL de uma página Web no *browser*. Isso é mostrado no lado direito da Figura 1.2. Primeiro, o computador enviará uma mensagem de requisição de conexão ao servidor Web e aguardará uma resposta. O servidor receberá essa mensagem de requisição de conexão e retornará uma mensagem de resposta de conexão. Sabendo que agora está tudo certo para requisitar o documento da Web, o computador envia então o nome da página Web que quer buscar naquele servidor com uma mensagem GET. Por fim, o servidor retorna a página (arquivo) para o computador.

Dados o exemplo humano e o exemplo de rede anteriores, as trocas de mensagens e as ações realizadas quando essas mensagens são enviadas e recebidas são os elementos fundamentais para a definição de um protocolo:

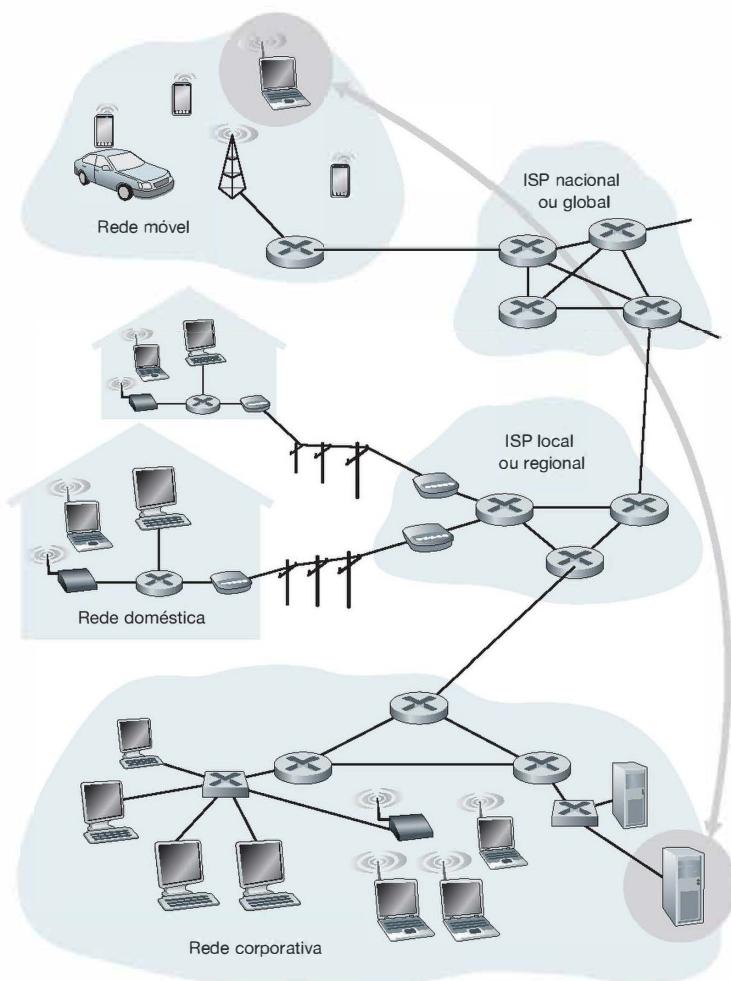
*Um **protocolo** define o formato e a ordem das mensagens trocadas entre duas ou mais entidades comunicantes, bem como as ações realizadas na transmissão e/ou no recebimento de uma mensagem ou outro evento.*

A Internet e as redes de computadores em geral fazem uso intenso de protocolos. Diferentes tipos são usados para realizar diferentes tarefas de comunicação. À medida que for avançando na leitura deste livro, você perceberá que alguns protocolos são simples e diretos, enquanto outros são complexos e intelectualmente profundos. Dominar a área de redes de computadores equivale a entender o que são, por que existem e como funcionam os protocolos de rede.

1.2 A PERIFERIA DA INTERNET

Nas seções anteriores, apresentamos uma descrição de alto nível da Internet e dos protocolos de rede. Agora passaremos a tratar com um pouco mais de profundidade os componentes de uma rede de computadores (e da Internet, em particular). Nesta seção, começamos pela periferia de uma rede e examinamos os componentes com os quais estamos mais familiarizados — a saber, computadores, smartphones e outros equipamentos que usamos diariamente. Na seção seguinte, passaremos da periferia para o núcleo da rede e estudaremos comutação e roteamento em redes de computadores.

Como descrito na seção anterior, no jargão de rede de computadores, os computadores e outros dispositivos conectados à Internet são frequentemente chamados de sistemas finais, pois se encontram na periferia da Internet, como mostrado na Figura 1.3. Os sistemas finais da Internet incluem computadores de mesa (por exemplo, PCs de mesa, MACs e caixas Linux), servidores (por exemplo, servidores Web e de e-mails), e computadores móveis (por exemplo, notebooks, smartphones e tablets). Além disso, diversos aparelhos alternativos estão sendo utilizados com a Internet como sistemas finais (veja nota em destaque).

Sistemas finais também são denominados *hospedeiros* (ou *hosts*) porque hospedam (isto é, executam) programas de aplicação, tais como um navegador (*browser*) da Web, um programa servidor da Web, um programa leitor de e-mail ou um servidor de e-mail. Neste livro, utilizaremos os termos hospedeiros e sistemas finais como sinônimos.

HISTÓRICO DO CASO

Um conjunto impressionante de sistemas finais da Internet

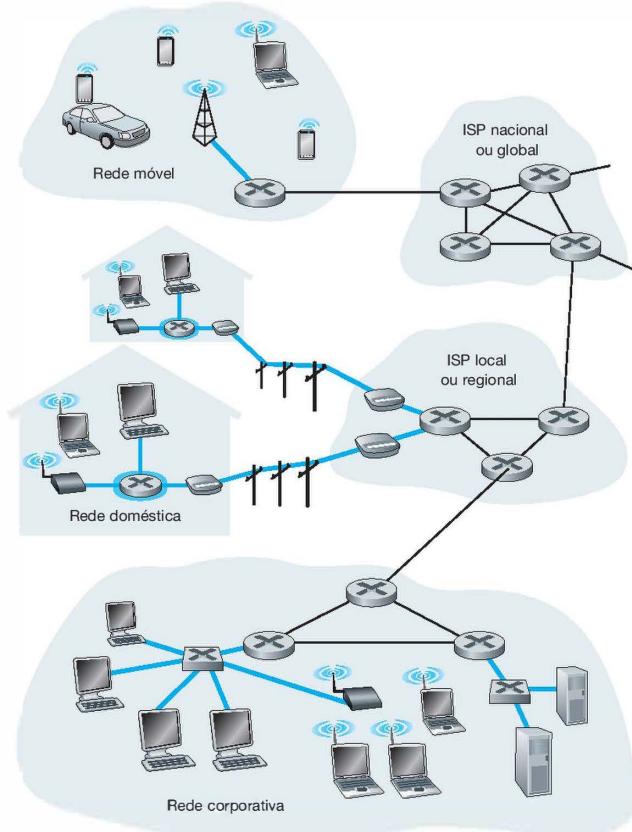
Não faz muito tempo, os sistemas finais conectados à Internet eram quase sempre computadores tradicionais, como máquinas de mesa e servidores de grande capacidade. Desde o final da década de 1990 até hoje, um amplo leque de equipamentos e dispositivos interessantes, cada vez mais diversos, vem sendo conectado à Internet, aproveitando sua capacidade de enviar e receber dados digitais. Tendo em vista a onipresença da Internet, seus protocolos bem definidos (padronizados) e a disponibilidade comercial de hardware capacitado para ela, é natural usar sua tecnologia para interconectar esses dispositivos entre si e a servidores conectados à Internet.

Muitos deles parecem ter sido criados exclusivamente para diversão — consoles de videogame (por exemplo, Xbox da Microsoft), televisores habilitados para Internet, quadros de fotos digitais que baixam e exibem imagens digitais, máquinas de lavar, refrigeradores e até mesmo uma torradeira da Internet que baixa informações meteorológicas de um servidor e grava uma imagem da previsão do tempo do dia em questão (por exemplo, nublado, com sol) na sua torrada matinal [BBC, 2001]. Telefones celulares que utilizam IP com recursos de GPS permitem o uso fácil de serviços dependentes do local (mapas, informações sobre serviços ou pessoas nas proximidades). Redes de sensores incorporadas ao ambiente físico permitem a monitoração de prédios, pontes, atividade sísmica, habitats da fauna selvagem, estuários de rios e clima. Aparelhos biomédicos podem ser incorporados e conectados em rede, numa espécie de rede corporal. Com tantos dispositivos diversificados sendo conectados em rede, a Internet está realmente se tornando uma "Internet de coisas" [ITU, 2005b].

FIGURA 1.3 INTERAÇÃO ENTRE SISTEMAS FINAIS

Às vezes, sistemas finais são ainda subdivididos em duas categorias: **clientes** e **servidores**. Informalmente, clientes costumam ser PCs de mesa ou portáteis, smartphones e assim por diante, ao passo que servidores tendem a ser máquinas mais poderosas, que armazenam e distribuem páginas Web, vídeo em tempo real, retransmissão de e-mails e assim por diante. Hoje, a maioria dos servidores dos quais recebemos resultados de busca, e-mail, páginas e vídeos reside em grandes **datacenters**. Por exemplo, o Google tem 30 a 50 datacenters, com muitos deles tendo mais de cem mil servidores.

1.2.1 Redes de acesso

Tendo considerado as aplicações e sistemas finais na "periferia da Internet", vamos agora considerar a rede de acesso — a rede física que conecta um sistema final ao primeiro roteador (também conhecido como "roteador

de borda") de um caminho partindo de um sistema final até outro qualquer. A Figura 1.4 apresenta diversos tipos de redes de acesso com linhas espessas, linhas cinzas e os ambientes (doméstico, corporativo e móvel sem fio) em que são usadas.

Acesso doméstico: DSL, cabo, FTTH, discado e satélite

Hoje, nos países desenvolvidos, mais de 65% dos lares possuem acesso à Internet, e, dentre eles, Coreia, Holanda, Finlândia e Suécia lideram com mais de 80%, quase todos por meio de uma conexão de banda larga em alta velocidade [ITU, 2011]. A Finlândia e a Espanha há pouco declararam que o acesso à Internet de alta velocidade é um "direito legal". Dado a esse interesse intenso no acesso doméstico, vamos começar nossa introdução às redes de acesso considerando como os lares se conectam à Internet.

Os dois tipos de acesso residencial banda largas predominantes são a **linha digital de assinante (DSL)** ou a cabo. Normalmente uma residência obtém acesso DSL à Internet da mesma empresa que fornece acesso telefônico local com fio (por exemplo, a operadora local). Assim, quando a DSL é utilizada, uma operadora do cliente é também seu provedor de serviços de Internet (ISP). Como ilustrado na Figura 1.5, o modem DSL de cada cliente utiliza a linha telefônica existente (par de fios de cobre trançado, que discutiremos na Seção 1.2.2) para trocar

FIGURA 1.4 REDES DE ACESSO

dados com um multiplexador digital de acesso à linha do assinante (DSLAM), em geral localizado na CT da operadora. O modem DSL da casa apanha dados digitais e os traduz para sons de alta frequência, para transmissão pelos fios de telefone até a CT; os sinais analógicos de muitas dessas residências são traduzidos de volta para o formato digital no DSLAM.

A linha telefônica conduz, simultaneamente, dados e sinais telefônicos tradicionais, que são codificados em frequências diferentes:

- um canal *downstream* de alta velocidade, com uma banda de 50 kHz a 1 MHZ;
- um canal *upstream* de velocidade média, com uma banda de 4 kHz a 50 kHz;
- um canal de telefone bidirecional comum, com uma banda de 0 a 4 kHz.

Essa abordagem faz que a conexão DSL pareça três conexões distintas, de modo que um telefonema e a conexão com a Internet podem compartilhar a DSL ao mesmo tempo. (Descreveremos essa técnica de multiplexação por divisão de frequência na Seção 1.3.2.) Do lado do consumidor, para os sinais que chegam até sua casa, um distribuidor separa os dados e os sinais telefônicos e conduz o sinal com os dados para o modem DSL. Na operadora, na CT, o DSLAM separa os dados e os sinais telefônicos e envia aqueles para a Internet. Centenas ou mesmo milhares de residências se conectam a um único DSLAM [Dischinger, 2007].

Os padrões DSL definem taxas de transmissão de 12 Mbits/s *downstream* e 1,8 Mbits/s *upstream* [ITU, 1999] e 24 Mbits/s *downstream* e 2,5 Mbits/s *upstream* [ITU, 2003]. Em razão de as taxas de transmissão e recebimento serem diferentes, o acesso é conhecido como assimétrico. As taxas reais alcançadas podem ser menores do que as indicadas anteriormente, pois o provedor de DSL pode, de modo proposital, limitar uma taxa residencial quando é oferecido o serviço em camadas (diferentes taxas, disponíveis a diferentes preços), ou porque a taxa máxima pode ser limitada pela distância entre a residência e a CT, pela bitola da linha de par trançado e pelo grau de interferência elétrica. Os engenheiros projetaram o DSL expressamente para distâncias curtas entre a residência e a CT; quase sempre, se a residência não estiver localizada dentro de 8 a 16 quilômetros da CT, ela precisa recorrer a uma forma de acesso alternativa à Internet.

Embora o DSL utilize a infraestrutura de telefone local da operadora, o **acesso à Internet a cabo** utiliza a infraestrutura de TV a cabo da operadora de televisão. Uma residência obtém acesso à Internet a cabo da mesma empresa que fornece a televisão a cabo. Como ilustrado na Figura 1.6, as fibras óticas conectam o terminal de distribuição às junções da região, sendo o cabo coaxial tradicional utilizado para chegar às casas e apartamentos de maneira individual. Cada junção costuma suportar de 500 a 5.000 casas. Em razão de a fibra e o cabo coaxial fazerem parte desse sistema, a rede é denominada híbrida fibra-coaxial (HFC).

O acesso à Internet a cabo necessita de modems especiais, denominados modems a cabo. Como o DSL, o modem a cabo é, em geral, um aparelho externo que se conecta ao computador residencial pela porta Ethernet. (Discutiremos Ethernet em detalhes no Capítulo 5.) No terminal de distribuição, o sistema de término do modem a cabo (CMTS) tem uma função semelhante à do DSLAM da rede DSL — transformar o sinal analógico

FIGURA 1.5 ACESSO À INTERNET POR DSL

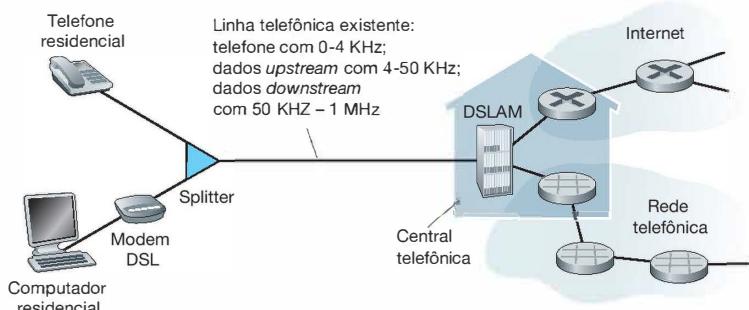
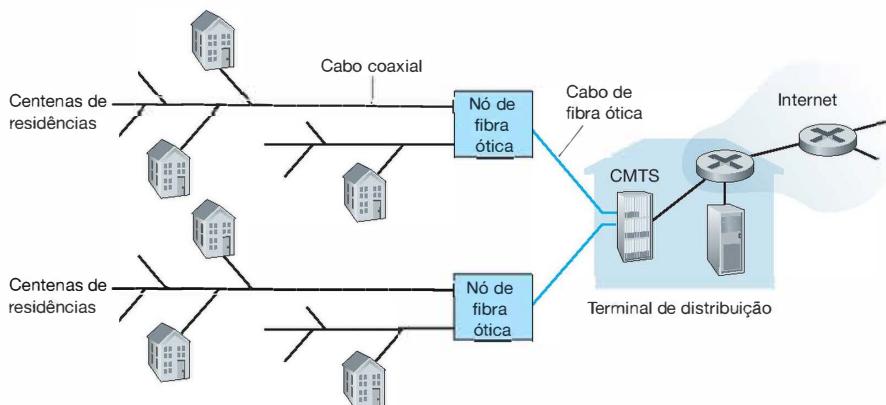
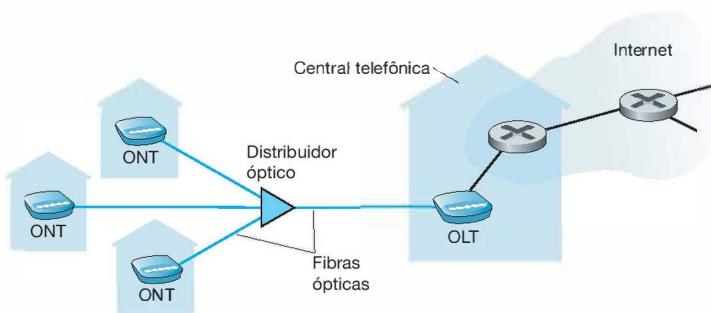



FIGURA 1.6 UMA REDE DE ACESSO HÍBRIDA FIBRA-COAXIAL

enviado dos modems a cabo de muitas residências *downstream* para o formato digital. Os modems a cabo dividem a rede HFC em dois canais, um de transmissão (*downstream*) e um de recebimento (*upstream*). Como a tecnologia DSL, o acesso costuma ser assimétrico, com o canal *downstream* recebendo uma taxa de transmissão maior do que a do canal *upstream*. O padrão DOCSIS 2.0 define taxas *downstream* de até 42,8 Mbits/s e taxas *upstream* de até 30,7 Mbits/s. Como no caso das redes DSL, a taxa máxima possível de ser alcançada pode não ser observada por causa de taxas de dados contratadas inferiores ou problemas na mídia.

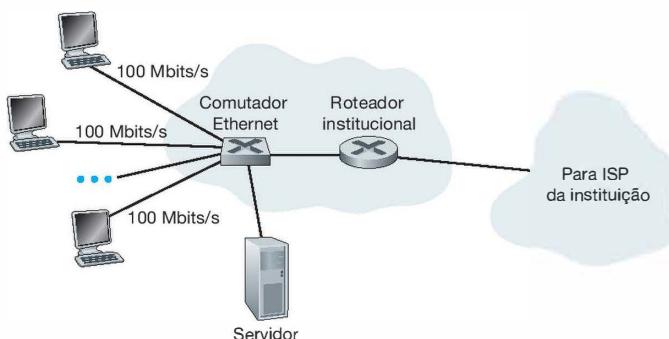
Uma característica importante do acesso a cabo é o fato de ser um meio de transmissão compartilhado. Em especial, cada pacote enviado pelo terminal viaja pelos enlaces *downstream* até cada residência e cada pacote enviado por uma residência percorre o canal *upstream* até o terminal de transmissão. Por essa razão, se diversos usuários estiverem fazendo o *download* de um arquivo em vídeo ao mesmo tempo no canal *downstream*, cada um receberá o arquivo a uma taxa bem menor do que a taxa de transmissão a cabo agregada. Por outro lado, se há somente alguns usuários ativos navegando, então cada um poderá receber páginas da Web a uma taxa de *downstream* máxima, pois esses usuários raramente solicitarão uma página ao mesmo tempo. Como o canal *upstream* também é compartilhado, é necessário um protocolo de acesso múltiplo distribuído para coordenar as transmissões e evitar colisões. (Discutiremos a questão de colisão no Capítulo 5.)

Embora as redes DSL e a cabo representem mais de 90% do acesso de banda larga residencial nos Estados Unidos, uma tecnologia que promete velocidades ainda mais altas é a implantação da ***fiber to the home*** **(FTTH)** [FTTH Council, 2011a]. Como o nome indica, o conceito da FTTH é simples — oferece um caminho de fibra ótica da CT diretamente até a residência. Nos Estados Unidos, a Verizon saiu na frente com a tecnologia FTTH, lançando o serviço FIOS [Verizon FIOS, 2012].

Existem várias tecnologias concorrentes para a distribuição ótica das CTs às residências. A rede mais simples é chamada fibra direta, para a qual existe uma fibra saindo da CT para cada casa. Em geral, uma fibra que sai da central telefônica é compartilhada por várias residências; ela é dividida em fibras individuais do cliente apenas após se aproximar relativamente das casas. Duas arquiteturas concorrentes de rede de distribuição ótica apresentam essa divisão: redes óticas ativas (AONs) e redes óticas passivas (PONs). A AON é na essência a Ethernet comutada, assunto discutido no Capítulo 5.

Aqui, falaremos de modo breve sobre a PON, que é utilizada no serviço FIOS da Verizon. A Figura 1.7 mostra a FTTH utilizando a arquitetura de distribuição de PON. Cada residência possui um terminal de rede ótica (ONT), que é conectado por uma fibra ótica dedicada a um distribuidor da região. O distribuidor combina certo número de residências (em geral menos de 100) a uma única fibra ótica compartilhada, que se liga a um terminal de linha ótica (OLT) na CT da operadora. O OLT, que fornece conversão entre sinais ópticos e elétricos, se co-

FIGURA 1.7 ACESSO A INTERNET POR FTTH

necta à Internet por meio de um roteador da operadora. Na residência, o usuário conecta ao ONT um roteador residencial (quase sempre sem fio) pelo qual acessa a Internet. Na arquitetura de PON, todos os pacotes enviados do OLT ao distribuidor são nele replicados (semelhante ao terminal de distribuição a cabo).

A FTTH consegue potencialmente oferecer taxas de acesso à Internet na faixa de gigabits por segundo. Porém, a maioria dos provedores de FTTH oferece diferentes taxas, das quais as mais altas custam muito mais. A velocidade de *downstream* média dos clientes FTTH nos Estados Unidos era de mais ou menos 20 Mbits/s em 2011 (em comparação com 13 Mbits/s para as redes de acesso a cabo e menos de 5 Mbits/s para DSL) [FTTH Council, 2011b].

Duas outras tecnologias também são usadas para oferecer acesso da residência à Internet. Em locais onde DSL, cabo e FTTH não estão disponíveis (por exemplo, em algumas propriedades rurais), um enlace de satélite pode ser empregado para conexão em velocidades não maiores do que 1 Mbit/s; StarBand e HughesNet são dois desses provedores de acesso por satélite. O acesso discado por linhas telefônicas tradicionais é baseado no mesmo modelo do DSL — um modem doméstico se conecta por uma linha telefônica a um modem no ISP. Em comparação com DSL e outras redes de acesso de banda larga, o acesso discado é terrivelmente lento em 56 kbits/s.

Acesso na empresa (e na residência): Ethernet e Wi-Fi

Nos *campi* universitários e corporativos, e cada vez mais em residências, uma rede local (LAN) costuma ser usada para conectar sistemas finais ao roteador da periferia. Embora existam muitos tipos de tecnologia LAN, a Ethernet é, de longe, a de acesso predominante nas redes universitárias, corporativas e domésticas. Como mostrado na Figura 1.8, os usuários utilizam um par de fios de cobre trançado para se conectarem a um comutador Ethernet, uma tecnologia tratada com mais detalhes no Capítulo 5. O comutador Ethernet, ou uma rede desses comutadores interconectados, é por sua vez conectado à Internet maior. Com o acesso por uma rede Ethernet, os usuários normalmente têm acesso de 100 Mbits/s com o comutador Ethernet, enquanto os servidores possuem um acesso de 1 Gbit/s ou até mesmo 10 Gbits/s.

Está cada vez mais comum as pessoas acessarem a Internet sem fio, seja por notebooks, smartphones, tablets ou por outros dispositivos (veja o texto "Um conjunto impressionante de sistemas finais da Internet", na seção "Histórico do caso", p. 9). Em uma LAN sem fio, os usuários transmitem/recebem pacotes para/de um ponto de acesso que está conectado à rede da empresa (quase sempre incluindo Ethernet com fio) que, por sua vez, é conectada à Internet com fio. Um usuário de LAN sem fio deve estar no espaço de alguns metros do ponto de acesso. O acesso à LAN sem fio baseado na tecnologia IEEE 802.11, ou seja, Wi-Fi, está presente em todo lugar — universidades, empresas, cafés, aeroportos, residências e, até mesmo, em aviões. Em muitas cidades, é possível ficar na esquina de uma rua e estar dentro da faixa de dez ou vinte estações-base (para um mapa global de estações-base 802.11 que foram descobertas e acessadas por pessoas que apreciam coisas do tipo, veja wigle.net [2012]). Como discutido com detalhes no Capítulo 6, hoje o 802.11 fornece uma taxa de transmissão compartilhada de até 54 Mbits/s.

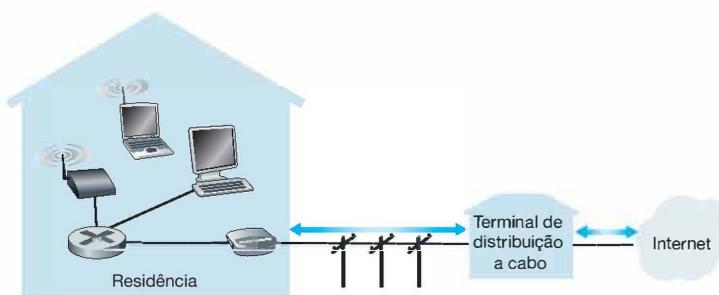



FIGURA 1.8 ACESSO A INTERNET POR ETHERNET

Embora as redes de acesso por Ethernet e Wi-Fi fossem implantadas no início em ambientes corporativos (empresas, universidades), elas há pouco se tornaram componentes bastante comuns das redes residenciais. Muitas casas unem o acesso residencial banda larga (ou seja, modems a cabo ou DSL) com a tecnologia LAN sem fio a um custo acessível para criar redes residenciais potentes [Edwards, 2011]. A Figura 1.9 mostra um esquema de uma rede doméstica típica. Ela consiste em um notebook móvel e um computador com fio; uma estação-base (o ponto de acesso sem fio), que se comunica com o computador sem fio; um modem a cabo, fornecendo acesso banda larga à Internet; e um roteador, que interconecta a estação-base e o computador fixo com o modem a cabo. Essa rede permite que os moradores tenham acesso banda larga à Internet com um usuário se movimentando da cozinha ao quintal e até os quartos.

Acesso sem fio em longa distância: 3G e LTE

Cada vez mais, dispositivos como iPhones, BlackBerrys e dispositivos Android estão sendo usados para enviar e-mail, navegar na Web, tuitar e baixar música enquanto se movimentam. Esses dispositivos empregam a mesma infraestrutura sem fios usada para a telefonia celular para enviar/receber pacotes por uma estação-base que é controlada pela operadora da rede celular. Diferente do Wi-Fi, um usuário só precisa estar dentro de algumas dezenas de quilômetros (ao contrário de algumas dezenas de metros) da estação-base.

As empresas de telecomunicação têm investido enormemente na assim chamada terceira geração (3G) sem fio, que oferece acesso remoto à Internet por pacotes comutados a velocidades que ultrapassam 1 Mbit/s. Porém, até mesmo tecnologias de acesso remotas de maior velocidade — uma quarta geração (4G) — já estão sendo im-

FIGURA 1.9 ESQUEMA DE UMA REDE DOMÉSTICA TÍPICA

plantadas. LTE (de "Long-Term Evolution", um candidato ao prêmio de Pior Acrônimo do Ano, PAA) tem suas raízes na tecnologia 3G, e tem potencial para alcançar velocidades superiores a 10 Mbits/s. Taxas *downstream* LTE de muitas dezenas de Mbits/s foram relatadas em implementações comerciais. Veremos os princípios básicos das redes sem fio e mobilidade, além de tecnologias Wi-Fi, 3G e LTE (e mais!) no Capítulo 6.

1.2.2 Meios físicos

Na subseção anterior, apresentamos uma visão geral de algumas das mais importantes tecnologias de acesso à Internet. Ao descrevê-las, indicamos também os meios físicos utilizados por elas. Por exemplo, dissemos que o HFC usa uma combinação de cabo de fibra ótica. Dissemos que DSL e Ethernet utilizam fios de cobre. Dissemos também que redes de acesso móveis usam o espectro de rádio. Nesta subseção damos uma visão geral desses e de outros meios de transmissão empregados na Internet.

Para definir o que significa meio físico, vamos pensar na curta vida de um bit. Considere um bit saindo de um sistema final, transitando por uma série de enlaces e roteadores e chegando a outro sistema final. Esse pobre e pequeno bit é transmitido muitas e muitas vezes. Primeiro, o sistema final originador transmite o bit e, logo em seguida, o primeiro roteador da série recebe-o; então, o primeiro roteador envia-o para o segundo roteador e assim por diante. Assim, nosso bit, ao viajar da origem ao destino, passa por uma série de pares transmissor--receptor, que o recebem por meio de ondas eletromagnéticas ou pulsos ópticos que se propagam por um **meio físico**. Com muitos aspectos e formas possíveis, o meio físico não precisa ser obrigatoriamente do mesmo tipo para cada par transmissor–receptor ao longo do caminho. Alguns exemplos de meios físicos são: par de fios de cobre trançado, cabo coaxial, cabo de fibra ótica multimodo, espectro de rádio terrestre e espectro de rádio por satélite. Os meios físicos se enquadram em duas categorias: **meios guiados** e **meios não guiados**. Nos meios guiados, as ondas são dirigidas ao longo de um meio sólido, tal como um cabo de fibra ótica, um par de fios de cobre trançado ou um cabo coaxial. Nos meios não guiados, as ondas se propagam na atmosfera e no espaço, como é o caso de uma LAN sem fio ou de um canal digital de satélite.

Contudo, antes de examinar as características dos vários tipos de meios, vamos discutir um pouco seus custos. O custo real de um enlace físico (fio de cobre, cabo de fibra ótica e assim por diante) costuma ser insignificante em comparação a outros. Em especial, o custo da mão de obra de instalação do enlace físico pode ser várias vezes maior do que o do material. Por essa razão, muitos construtores instalam pares de fios trançados, fibra ótica e cabo coaxial em todas as salas de um edifício. Mesmo que apenas um dos meios seja usado inicialmente, há uma boa probabilidade de outro ser usado no futuro próximo — portanto, poupa-se dinheiro por não ser preciso instalar fiação adicional depois.

Par de fios de cobre trançado

O meio de transmissão guiado mais barato e mais usado é o par de fios de cobre trançado, que vem sendo empregado há mais de cem anos nas redes de telefonia. De fato, mais de 99% da fiação que conecta aparelhos telefônicos a centrais locais utilizam esse meio. Quase todos nós já vimos um em casa ou no local de trabalho: esse par constituído de dois fios de cobre isolados, cada um com cerca de um milímetro de espessura, enrolados em espiral. Os fios são trançados para reduzir a interferência elétrica de pares semelhantes que estejam próximos. Normalmente, uma série de pares é conjugada dentro de um cabo, isolando-se os pares com blindagem de proteção. Um par de fios constitui um único enlace de comunicação. O **par trançado sem blindagem** (*unshielded twisted pair* — UTP) costuma ser usado em redes de computadores de edifícios, isto é, em LANs. Hoje, as taxas de transmissão de dados para as LANs de pares trançados estão na faixa de 10 Mbits/s a 10 Gbits/s. As taxas de transmissão de dados que podem ser alcançadas dependem da bitola do fio e da distância entre transmissor e receptor.

Quando a tecnologia da fibra ótica surgiu na década de 1980, muitos depreciaram o par trançado por suas taxas de transmissão de bits relativamente baixas. Alguns até acharam que a tecnologia da fibra ótica o substituiria por completo. Mas ele não desistiu assim tão facilmente. A moderna tecnologia de par trançado, tal como o

cabo de categoria 6a, pode alcançar taxas de transmissão de dados de 10 Gbits/s para distâncias de até algumas centenas de metros. No final, o par trançado firmou-se como a solução dominante para LANs de alta velocidade.

Como vimos anteriormente, o par trançado também é usado para acesso residencial à Internet. Vimos que a tecnologia do modem discado possibilita taxas de acesso de até 56 kbits/s com pares trançados. Vimos também que a tecnologia DSL (linha digital de assinante) permitiu que usuários residenciais acessem a Internet em dezenas de Mbits/s com pares de fios trançados (quando as residências estão próximas ao modem do ISP).

Cabo coaxial

Como o par trançado, o cabo coaxial é constituído de dois condutores de cobre, porém concêntricos e não paralelos. Com essa configuração, isolamento e blindagem especiais, pode alcançar taxas altas de transmissão de dados. Cabos coaxiais são muito comuns em sistemas de televisão a cabo. Como já comentamos, recentemente sistemas de televisão a cabo foram acoplados com modems a cabo para oferecer aos usuários residenciais acesso à Internet a velocidades de dezenas de Mbits/s. Em televisão a cabo e acesso a cabo à Internet, o transmissor passa o sinal digital para uma banda de frequência específica e o sinal analógico resultante é enviado do transmissor para um ou mais receptores. O cabo coaxial pode ser utilizado como um **meio compartilhado** guiado. Vários sistemas finais podem ser conectados diretamente ao cabo, e todos eles recebem qualquer sinal que seja enviado pelos outros sistemas finais.

Fibras ópticas

A fibra ótica é um meio delgado e flexível que conduz pulsos de luz, cada um deles representando um bit. Uma única fibra ótica pode suportar taxas de transmissão elevadíssimas, de até dezenas ou mesmo centenas de gigabits por segundo. Fibras óticas são imunes à interferência eletromagnética, têm baixíssima atenuação de sinal até cem quilômetros e são muito difíceis de derivar. Essas características fizeram da fibra ótica o meio preferido para a transmissão guiada de grande alcance, em especial para cabos submarinos. Hoje, muitas redes telefônicas de longa distância dos Estados Unidos e de outros países usam exclusivamente fibras óticas, que também predominam no *backbone* da Internet. Contudo, o alto custo de equipamentos ópticos — como transmissores, receptores e comutadores — vem impedindo sua utilização para transporte a curta distância, como em LANs ou em redes de acesso residenciais. As velocidades de conexão do padrão Optical Carrier (OC) variam de 51,8 Mbits/s a 39,8 Gbits/s; essas especificações são frequentemente denominadas OC-*n*, em que a velocidade de conexão se iguala a $n \times 51{,}8$ Mbits/s. Os padrões usados hoje incluem OC-1, OC-3, OC-12, OC-24, OC-48, OC-96, OC-192 e OC-768. Mukherjee [2006] e Ramaswamy [2010] apresentam uma abordagem de vários aspectos da rede óptica.

Canais de rádio terrestres

Canais de rádio carregam sinais dentro do espectro eletromagnético. São um meio atraente porque sua instalação não requer cabos físicos, podem atravessar paredes, dão conectividade ao usuário móvel e, potencialmente, conseguem transmitir um sinal a longas distâncias. As características de um canal de rádio dependem muito do ambiente de propagação e da distância pela qual o sinal deve ser transmitido. Condições ambientais determinam perda de sinal no caminho e atenuação por efeito de sombra (que reduz a intensidade do sinal quando ele transita por distâncias longas e ao redor/através de objetos interferentes), atenuação por caminhos múltiplos (devido à reflexão do sinal quando atinge objetos interferentes) e interferência (por outras transmissões ou sinais eletromagnéticos).

Canais de rádio terrestres podem ser classificados, de modo geral, em três grupos: os que operam sobre distâncias muito curtas (por exemplo, com um ou dois metros); os de pequeno alcance, que funcionam em locais próximos, normalmente abrangendo de dez a algumas centenas de metros, e os de longo alcance, que abrangem dezenas de quilômetros. Dispositivos pessoais como fones sem fio, teclados e dispositivos médicos operam por curtas distâncias; as tecnologias LAN sem fio, descritas na Seção 1.2.1, utilizam canais de rádio local; as tecnolo-

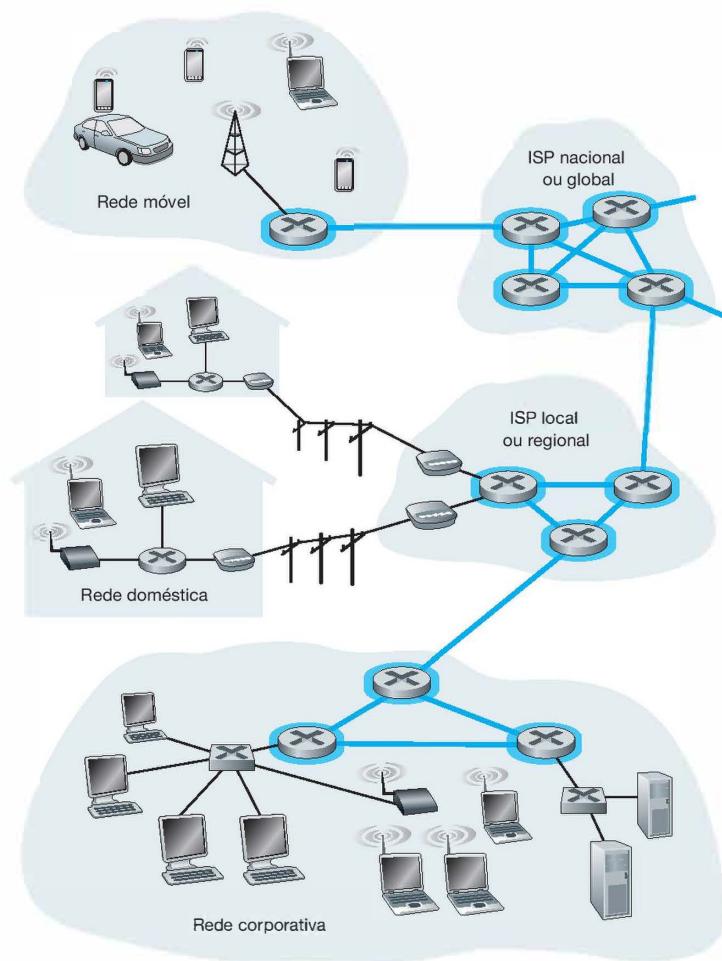
gias de acesso em telefone celular utilizam canal de rádio de longo alcance. Abordaremos canais de rádio detalhadamente no Capítulo 6.

Canais de rádio por satélite

Um satélite de comunicação liga dois ou mais transmissores-receptores de micro-ondas baseados na Terra, denominados estações terrestres. Ele recebe transmissões em uma faixa de frequência, gera novamente o sinal usando um repetidor (sobre o qual falaremos a seguir) e o transmite em outra frequência. Dois tipos de satélites são usados para comunicações: **satélites geoestacionários** e **satélites de órbita baixa (LEO)**.

Os satélites geoestacionários ficam de modo permanente sobre o mesmo lugar da Terra. Essa presença estacionária é conseguida colocando-se o satélite em órbita a 36 mil quilômetros acima da superfície terrestre. Essa enorme distância da estação terrestre ao satélite e de seu caminho de volta à estação terrestre traz um substancial atraso de propagação de sinal de 280 milissegundos. Mesmo assim, enlaces por satélite, que podem funcionar a velocidades de centenas de Mbits/s, são frequentemente usados em áreas sem acesso à Internet baseado em DSL ou cabo.

Os satélites de órbita baixa são posicionados muito mais próximos da Terra e não ficam sempre sobre um único lugar. Eles giram ao redor da Terra (exatamente como a Lua) e podem se comunicar uns com os outros e com estações terrestres. Para prover cobertura contínua em determinada área, é preciso colocar muitos satélites em órbita. Hoje, existem muitos sistemas de comunicação de baixa altitude em desenvolvimento. A página da Web referente à constelação de satélites da Lloyd [Wood, 2012] fornece e coleta informações sobre esses sistemas para comunicações. A tecnologia de satélites de órbita baixa poderá ser utilizada para acesso à Internet no futuro.

1.3 O NÚCLEO DA REDE

Após termos examinado a periferia da Internet, vamos agora nos aprofundar mais no núcleo da rede — a rede de comutadores de pacote e enlaces que interconectam os sistemas finais da Internet. Os núcleos da rede aparecem destacados em cinza na Figura 1.10.

1.3.1 Comutação de pacotes

Em uma aplicação de rede, sistemas finais trocam **mensagens** entre si. Mensagens podem conter qualquer coisa que o projetista do protocolo queira. Podem desempenhar uma função de controle (por exemplo, as mensagens "oi" no nosso exemplo de comunicação na Figura 1.2) ou conter dados, tal como um e-mail, uma imagem JPEG ou um arquivo de áudio MP3. Para enviar uma mensagem de um sistema final de origem para um destino, o originador fragmenta mensagens longas em porções de dados menores, denominadas **pacotes**. Entre origem e destino, cada um deles percorre enlaces de comunicação e **comutadores de pacotes** (há dois tipos principais de comutadores de pacotes: **roteadores** e **comutadores de camada de enlace**). Pacotes são transmitidos por cada enlace de comunicação a uma taxa igual à de transmissão *total*. Assim, se um sistema final de origem ou um comutador de pacotes estiver enviando um pacote de L bits por um enlace com taxa de transmissão de R bits/s, então o tempo para transmitir o pacote é L/R segundos.

Transmissão armazena-e-reenvia

A maioria dos comutadores de pacotes utiliza a **transmissão armazena-e-reenvia (*store-and-forward*)** nas entradas dos enlaces. A transmissão armazena-e-reenvia significa que o comutador de pacotes deve receber o pacote inteiro antes de poder começar a transmitir o primeiro bit para o enlace de saída. Para explorar a transmissão armazena-e-reenvia com mais detalhes, considere uma rede simples, consistindo em dois sistemas finais conectados por um único roteador, conforme mostra a Figura 1.11. Um roteador em geral terá muitos enlaces incidentes, pois sua

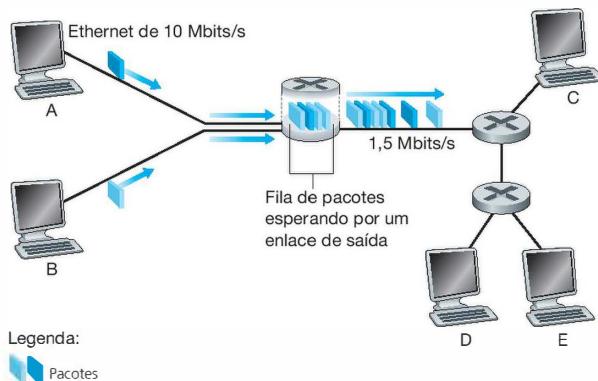
FIGURA 1.10 O NÚCLEO DA REDE

função é comutar um pacote que chega para um enlace de saída; neste exemplo simples, o roteador tem a tarefa de transferir um pacote de um enlace (entrada) para o único outro enlace conectado. Aqui, a origem tem três pacotes, cada um consistindo em L bits, para enviar ao destino. No instante de tempo mostrado na Figura 1.11, a origem transmitiu parte do pacote 1, e a frente do pacote 1 já chegou no roteador. Como emprega a transmissão armazena-e-reenvia, nesse momento, o roteador não pode transmitir os bits que recebeu; em vez disso, ele precisa primeiro manter em buffer (isto é, "armazenar") os bits do pacote. Somente depois que o roteador tiver recebido *todos* os bits, poderá começar a transmitir (isto é, "reenviar") o pacote para o enlace de saída. Para ter uma ideia da transmissão armazena-e-reenvia, vamos agora calcular a quantidade de tempo decorrido desde quando a origem começa a enviar até que o destino tenha recebido o pacote inteiro. (Aqui, ignoraremos o atraso de propagação — o tempo gasto para os bits atravessarem o fio em uma velocidade próxima à da luz —, o que será discutido na Seção 1.4.) A origem começa a transmitir no tempo 0; no tempo L/R segundos, a origem terá transmitido o pacote inteiro, que terá sido recebido e armazenado no roteador (pois não há atraso de propagação). No tempo L/R segundos, como o roteador

FIGURA 1.11 COMUTAÇÃO DE PACOTES ARMAZENA-E-REENVIA

já terá recebido o pacote inteiro, ele pode começar a transmiti-lo para o enlace de saída, em direção ao destino; no tempo $2L/R$, o roteador terá transmitido o pacote inteiro, e este terá sido recebido pelo destino. Assim, o atraso total é $2L/R$. Se o comutador, em vez disso, reenviasse os bits assim que chegassem (sem primeiro receber o pacote inteiro), então o atraso total seria L/R, pois os bits não são mantidos no roteador. Mas, conforme discutiremos na Seção 1.4, os roteadores precisam receber, armazenar e *processar* o pacote inteiro antes de encaminhar.

Agora vamos calcular a quantidade de tempo decorrido desde quando a origem começa a enviar o primeiro pacote até que o destino tenha recebido todos os três. Como antes, no instante L/R, o roteador começa a reenviar o primeiro pacote. Mas, também no tempo L/R, a origem começará a enviar o segundo, pois ela terá acabado de mandar o primeiro pacote inteiro. Assim, no tempo $2L/R$, o destino terá recebido o primeiro pacote e o roteador terá recebido o segundo. De modo semelhante, no instante $3L/R$, o destino terá recebido os dois primeiros pacotes e o roteador terá recebido o terceiro. Por fim, no tempo $4L/R$, o destino terá recebido todos os três pacotes!

Vamos considerar o caso geral do envio de um pacote da origem ao destino por um caminho que consiste em N enlaces, cada um com taxa R (assim, há $N - 1$ roteadores entre origem e destino). Aplicando a mesma lógica usada anteriormente, vemos que o atraso fim a fim é:

$$d_{\text{fim a fim}} = N\frac{L}{R} \tag{1.1}$$

Você poderá tentar determinar qual seria o atraso para P pacotes enviados por uma série de N enlaces.

Atrasos de fila e perda de pacote

A cada comutador de pacotes estão ligados vários enlaces. Para cada um destes, o comutador de pacotes tem um **buffer de saída** (também denominado **fila de saída**), que armazena pacotes prestes a serem enviados pelo roteador para aquele enlace. Os buffers de saída desempenham um papel fundamental na comutação de pacotes. Se um pacote que está chegando precisa ser transmitido por um enlace, mas o encontra ocupado com a transmissão de outro pacote, deve aguardar no buffer de saída. Desse modo, além dos atrasos de armazenagem e reenvio, os pacotes sofrem **atrasos de fila** no buffer de saída. Esses atrasos são variáveis e dependem do grau de congestionamento da rede. Como o espaço do buffer é finito, um pacote que está chegando pode encontrá-lo lotado de outros que estão esperando transmissão. Nesse caso, ocorrerá uma **perda de pacote** — um pacote que está chegando ou um dos que já estão na fila é descartado.

A Figura 1.12 ilustra uma rede simples de comutação de pacotes. Como na Figura 1.11, os pacotes são representados por placas tridimensionais. A largura de uma placa representa o número de bits no pacote. Nessa figura, todos os pacotes têm a mesma largura, portanto, o mesmo tamanho. Suponha que os hospedeiros A e B estejam enviando pacotes ao hospedeiro E. Os hospedeiros A e B primeiro enviarão seus pacotes por enlaces Ethernet de 10 Mbits/s até o primeiro comutador, que vai direcioná-los para o enlace de 1,5 Mbits/s. Se, durante um pequeno intervalo de tempo, a taxa de chegada de pacotes ao roteador (quando convertida para bits por segundo) for maior do que 1,5 Mbits/s, ocorrerá congestionamento no roteador, pois os pacotes formarão uma fila no buffer de saída do enlace antes de ser transmitidos para o enlace. Por exemplo, se cada um dos hospedeiros A e B enviar uma rajada de cinco pacotes de ponta a ponta ao mesmo tempo, então a maior parte deles gastará algum tempo

FIGURA 1.12 COMUTAÇÃO DE PACOTES

esperando na fila. De fato, a situação é semelhante a muitas no dia a dia — por exemplo, quando aguardamos na fila de um caixa de banco ou quando esperamos em uma cabine de pedágio. Vamos analisar esse atraso de fila mais detalhadamente na Seção 1.4.

Tabelas de repasse e protocolos de roteamento

Dissemos anteriormente que um roteador conduz um pacote que chega a um de seus enlaces de comunicação para outro de seus enlaces de comunicação conectados. Mas como o roteador determina o enlace que deve conduzir o pacote? Na verdade, isso é feito de diversas maneiras por diferentes tipos de rede de computadores. Aqui, descreveremos de modo resumido como isso é feito pela Internet.

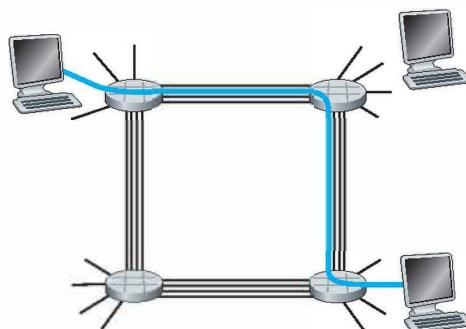
Na Internet, cada sistema final tem um endereço denominado endereço IP. Quando um sistema final de origem quer enviar um pacote a um destino, a origem inclui o endereço IP do destino no cabeçalho do pacote. Como os endereços postais, este possui uma estrutura hierárquica. Quando um pacote chega a um roteador na rede, este examina uma parte do endereço de destino e o conduz a um roteador adjacente. Mais especificamente, cada roteador possui uma **tabela de encaminhamento** que mapeia os endereços de destino (ou partes deles) para enlaces de saída desse roteador. Quando um pacote chega a um roteador, este examina o endereço e pesquisa sua tabela de encaminhamento, utilizando esse endereço de destino para encontrar o enlace de saída apropriado. O roteador, então, direciona o pacote a esse enlace de saída.

O processo de roteamento fim a fim é semelhante a um motorista que não quer consultar o mapa, preferindo pedir informações. Por exemplo, suponha que Joe vai dirigir da Filadélfia para 156 Lakeside Drive, em Orlando, Flórida. Primeiro, Joe vai ao posto de gasolina de seu bairro e pergunta como chegar a 156 Lakeside Drive, em Orlando, Flórida. O frentista do posto extrai a palavra Flórida do endereço e diz que Joe precisa pegar a interestadual I-95 South, cuja entrada fica ao lado do posto. Ele também diz a Joe para pedir outras informações assim que chegar à Flórida. Então, Joe pega a I-95 South até chegar a Jacksonville, na Flórida, onde pede mais informações a outro frentista. Este extrai a palavra Orlando do endereço a diz a Joe para continuar na I-95 até Daytona Beach, e lá se informar de novo. Em Daytona Beach, outro frentista também extrai a palavra Orlando do endereço e pede para que ele pegue a I-4 diretamente para Orlando. Joe segue suas orientações e chega a uma saída para Orlando. Ele vai até outro posto de gasolina, e dessa vez o frentista extrai a palavra Lakeside Drive do endereço e diz a ele qual estrada seguir para Lakeside Drive. Assim que Joe chega a Lakeside Drive, pergunta a uma criança andando de bicicleta como chegar a seu destino. A criança extrai o número 156 do endereço e aponta para a casa. Joe finalmente chega enfim a seu destino. Nessa analogia, os frentistas de posto de gasolina e as crianças andando de bicicleta são semelhantes aos roteadores.

Vimos que um roteador usa um endereço de destino do pacote para indexar uma tabela de encaminhamento e determinar o enlace de saída apropriado. Mas essa afirmação traz ainda outra questão: como as tabelas de encaminhamento são montadas? Elas são configuradas manualmente em cada roteador ou a Internet utiliza um procedimento mais automatizado? Essa questão será estudada com mais profundidade no Capítulo 4. Mas, para aguçar seu apetite, observe que a Internet possui uma série de **protocolos de roteamento** especiais, que são utilizados para configurar automaticamente as tabelas de encaminhamento. Um protocolo de roteamento pode, por exemplo, determinar o caminho mais curto de cada roteador a cada destino e utilizar os resultados para configurar as tabelas de encaminhamento nos roteadores.

Você gostaria de ver a rota fim a fim que os pacotes realizam na Internet? Nós o convidamos a colocar a mão na massa e interagir com o programa Traceroute, visitando o site <www.traceroute.org>, escolhendo uma origem em um país qualquer e traçando a rota dessa origem até o seu computador. (Para obter detalhes sobre o Traceroute, veja a Seção 1.4.)

1.3.2 Comutação de circuitos

Há duas abordagens fundamentais para locomoção de dados através de uma rede de enlaces e comutadores: **comutação de circuitos** e **comutação de pacotes**. Tendo visto estas últimas na subseção anterior, agora vamos voltar nossa atenção às redes de comutação de circuitos.

Nessas redes, os recursos necessários ao longo de um caminho (buffers, taxa de transmissão de enlaces) para oferecer comunicação entre os sistemas finais são *reservados* pelo período da sessão de comunicação entre os sistemas finais. Em redes de comutação de pacotes, tais recursos *não* são reservados; as mensagens de uma sessão usam os recursos por demanda e, como consequência, poderão ter de esperar (isto é, entrar na fila) para conseguir acesso a um enlace de comunicação. Como simples analogia, considere dois restaurantes — um que exige e outro que não exige nem aceita reserva. Se quisermos ir ao restaurante que exige reserva, teremos de passar pelo aborrecimento de telefonar antes de sair de casa. Mas, quando chegarmos lá, poderemos, em princípio, ser logo atendidos e servidos. No segundo restaurante, não precisaremos nos dar ao trabalho de reservar mesa, porém, quando lá chegarmos, talvez tenhamos de esperar para sentar.

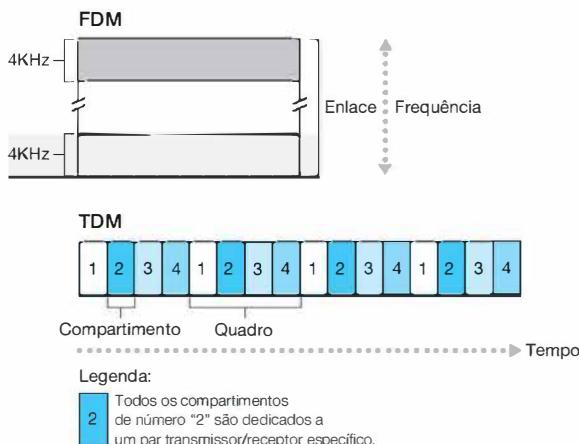
As redes de telefonia tradicionais são exemplos de redes de comutação de circuitos. Considere o que acontece quando uma pessoa quer enviar a outra uma informação (por voz ou por fax) por meio de uma rede telefônica. Antes que o remetente possa enviar a informação, a rede precisa estabelecer uma conexão entre ele e o destinatário. Essa é uma conexão forte, na qual os comutadores no caminho entre o remetente e o destinatário mantêm o estado. No jargão da telefonia, essa conexão é denominada **circuito**. Quando a rede estabelece o circuito, também reserva uma taxa de transmissão constante nos enlaces da rede durante o período da conexão. Visto que foi reservada largura de banda para essa conexão remetente-destinatário, o remetente pode transferir dados ao destinatário a uma taxa constante *garantida*.

A Figura 1.13 ilustra uma rede de comutação de circuitos. Nela, os quatro comutadores de circuitos estão interconectados por quatro enlaces. Cada enlace tem quatro circuitos, de modo que cada um pode suportar quatro conexões simultâneas. Cada um dos hospedeiros (por exemplo, PCs e estações de trabalho) está conectado diretamente a um dos circuitos. Quando dois sistemas finais querem se comunicar, a rede estabelece uma **conexão fim a fim** dedicada entre os dois hospedeiros. Assim, para que o sistema final A envie mensagens ao sistema final B, a rede deve primeiro reservar um circuito em cada um dos dois enlaces. Neste exemplo, a conexão fim a fim dedicada usa o segundo circuito no primeiro enlace e o quarto circuito no segundo enlace. Como cada enlace tem quatro circuitos, para cada enlace usado pela conexão fim a fim, esta fica com um quarto da capacidade de transmissão total durante o período da conexão. Assim, por exemplo, se cada enlace entre comutadores adjacentes tiver uma taxa de transmissão de 1 Mbit/s, então cada conexão de comutação de circuitos fim a fim obtém 250 kbits/s de taxa de transmissão dedicada.

Ao contrário, considere o que ocorre quando um sistema final quer enviar um pacote a outro hospedeiro por uma rede de comutação de pacotes, como a Internet. Como acontece na comutação de circuitos, o pacote é transmitido por uma série de enlaces de comunicação. Mas, na comutação de pacotes, ele é enviado à rede sem reservar qualquer

FIGURA 1.13 UMA REDE SIMPLES DE COMUTAÇÃO DE CIRCUITOS COMPOSTA DE QUATRO COMUTADORES E QUATRO ENLACES

recurso do enlace. Se um dos enlaces estiver congestionado porque outros pacotes precisam ser transmitidos ao mesmo tempo, então nosso pacote terá de esperar em um buffer na extremidade de origem do enlace de transmissão e sofrerá um atraso. A Internet faz o melhor esforço para entregar os dados de pronto, mas não dá garantia alguma.

Multiplexação em redes de comutação de circuitos

Um circuito é implementado em um enlace por **multiplexação por divisão de frequência** (*frequency-division multiplexing* — FDM) ou por **multiplexação por divisão de tempo** (*time-division multiplexing* — TDM). Com FDM, o espectro de frequência de um enlace é compartilhado entre as conexões estabelecidas através desse enlace. Ou seja, o enlace reserva uma banda de frequência para cada conexão durante o período da ligação. Em redes telefônicas, a largura dessa banda de frequência em geral é 4 kHz (isto é, 4 mil Hertz ou 4 mil ciclos por segundo). A largura da banda é denominada, claro, **largura de banda**. Estações de rádio FM também usam FDM para compartilhar o espectro de frequência entre 88 MHz e 108 MHz, sendo atribuída para cada estação uma banda de frequência específica.

Em um enlace TDM, o tempo é dividido em quadros de duração fixa, e cada quadro é dividido em um número fixo de compartimentos (*slots*). Quando estabelece uma conexão por meio de um enlace, a rede dedica à conexão um compartimento de tempo em cada quadro. Esses compartimentos são reservados para o uso exclusivo dessa conexão, e um dos compartimentos de tempo (em cada quadro) fica disponível para transmitir os dados dela.

A Figura 1.14 ilustra as técnicas FDM e TDM para um enlace de rede que suporta até quatro circuitos. Para FDM, o domínio de frequência é segmentado em quatro faixas, com largura de banda de 4 kHz cada. Para TDM, o domínio de tempo é segmentado em quadros, cada um com quatro compartimentos de tempo; a cada circuito é designado o mesmo compartimento dedicado nos quadros sucessivos TDM. Para TDM, a taxa de transmissão de um circuito é igual à taxa do quadro multiplicada pelo número de bits em um compartimento. Por exemplo, se o enlace transmite 8 mil quadros por segundo e cada compartimento consiste em 8 bits, então a taxa de transmissão de um circuito é 64 kbits/s.

Os defensores da comutação de pacotes sempre argumentaram que comutação de circuitos é desperdício, porque os circuitos dedicados ficam ociosos durante **períodos de silêncio**. Por exemplo, quando um dos participantes de uma conversa telefônica para de falar, os recursos ociosos da rede (bandas de frequências ou compartimentos nos enlaces ao longo da rota da conexão) não podem ser usados por outras conexões em curso. Para outro exemplo de como esses recursos podem ser subutilizados, considere um radiologista que usa uma rede de comutação de circuitos para acessar remotamente uma série de exames. Ele estabelece uma conexão, requisita uma imagem, examina-a e, em seguida, solicita uma nova. Recursos de rede são atribuídos à conexão, mas não utilizados (isto é, são desperdiçados) no período em que o radiologista examina a imagem. Defensores

FIGURA 1.14 COM FDM, CADA CIRCUITO DISPÕE CONTINUAMENTE DE UMA FRAÇÃO DA LARGURA DE BANDA. COM TDM, CADA CIRCUITO DISPÕE DE TODA A LARGURA DE BANDA PERIODICAMENTE, DURANTE BREVES INTERVALOS DE TEMPO (ISTO É, DURANTE COMPARTIMENTOS DE TEMPO)

da comutação de pacotes também gostam de destacar que estabelecer circuitos e reservar larguras de banda fim a fim é complicado e exige softwares complexos de sinalização para coordenar a operação dos comutadores ao longo do caminho.

Antes de encerrarmos esta discussão sobre comutação de circuitos, examinaremos um exemplo numérico que deverá esclarecer melhor o assunto. Vamos considerar o tempo que levamos para enviar um arquivo de 640 kbits/s do hospedeiro A ao hospedeiro B por uma rede de comutação de circuitos. Suponha que todos os enlaces da rede usem TDM de 24 compartimentos e tenham uma taxa de 1,536 Mbits/s. Suponha também que um circuito fim a fim leva 500 milissegundos para ser ativado antes que A possa começar a transmitir o arquivo. Em quanto tempo o arquivo será enviado? Cada circuito tem uma taxa de transmissão de (1,536 Mbits/s)/24 = 64 kbits/s; portanto, demorará (640 kbits/s)/(64 kbits/s) = 10 segundos para transmitir o arquivo. A esses 10 segundos adicionamos o tempo de ativação do circuito, resultando 10,5 segundos para o envio. Observe que o tempo de transmissão é independente do número de enlaces: o tempo de transmissão seria 10 segundos se o circuito fim a fim passasse por um ou por uma centena de enlaces. (O atraso real fim a fim também inclui um atraso de propagação; ver Seção 1.4.)

Comutação de pacotes *versus* comutação de circuitos

Agora que já descrevemos comutação de pacotes e comutação de circuitos, vamos comparar as duas. Opositores da comutação de pacotes costumam argumentar que ela não é adequada para serviços de tempo real (por exemplo, ligações telefônicas e videoconferência) por causa de seus atrasos fim a fim variáveis e imprevisíveis (que se devem principalmente a variáveis e imprevisíveis atrasos de fila). Defensores da comutação de pacotes argumentam que (1) ela oferece melhor compartilhamento de banda do que comutação de circuitos e (2) sua implementação é mais simples, mais eficiente e mais barata do que a de comutação de circuitos. Uma discussão interessante sobre comutação de pacotes e comutação de circuitos pode ser encontrada em Molinero-Fernandez [2002]. De modo geral, quem não gosta de perder tempo fazendo reserva de mesa em restaurantes prefere comutação de pacotes à comutação de circuitos.

Por que a comutação de pacotes é mais eficiente? Vamos examinar um exemplo simples. Suponha que usuários compartilhem um enlace de 1 Mbit/s. Considere também que cada usuário alterne períodos de atividade, quando gera dados a uma taxa constante de 100 kbits/s, e de inatividade, quando não gera dados. Imagine ainda

que o usuário esteja ativo apenas 10% do tempo (e fique ocioso, tomando cafezinho, durante os restantes 90%). Com comutação de circuitos, devem ser *reservados* 100 kbits/s para *cada usuário* durante todo o tempo. Por exemplo, com TDM, se um quadro de um segundo for dividido em 10 compartimentos de tempo de 100 milissegundos cada, então seria alocado um compartimento de tempo por quadro a cada usuário.

Desse modo, o enlace de comutação de circuitos pode suportar somente 10 (= 1 Mbit/s/100 kbits/s) usuários simultaneamente. Com a comutação de pacotes, a probabilidade de haver um usuário específico ativo é 0,1 (isto é, 10%). Se houver 35 usuários, a probabilidade de haver 11 ou mais usuários ativos ao mesmo tempo é de mais ou menos 0,0004. (O Problema P8 dos Exercícios de Fixação demonstra como essa probabilidade é calculada.) Quando houver dez ou menos usuários ativos simultâneos (a probabilidade de isso acontecer é 0,9996), a taxa agregada de chegada de dados é menor ou igual a 1 Mbit/s, que é a taxa de saída do enlace. Assim, quando houver dez ou menos usuários ativos, pacotes de usuários fluirão pelo enlace essencialmente sem atraso, como é o caso na comutação de circuitos. Quando houver mais de dez usuários ativos ao mesmo tempo, a taxa agregada de chegada de pacotes excederá a capacidade de saída do enlace, e a fila de saída começará a crescer. (E continuará a crescer até que a velocidade agregada de entrada caia novamente para menos de 1 Mbit/s, ponto em que o comprimento da fila começará a diminuir.) Como a probabilidade de haver mais de dez usuários ativos é ínfima nesse exemplo, a comutação de pacotes apresenta, em essência, o mesmo desempenho da comutação de circuitos, *mas o faz para mais de três vezes o número de usuários.*

Vamos considerar agora um segundo exemplo simples. Suponha que haja dez usuários e que um deles de repente gere mil pacotes de mil bits, enquanto os outros nove permanecem inativos e não geram pacotes. Com comutação de circuitos TDM de dez compartimentos de tempo por quadro, e cada quadro consistindo em mil bits, o usuário ativo poderá usar somente seu único compartimento por quadro para transmitir dados, enquanto os nove compartimentos restantes em cada quadro continuarão ociosos. Dez segundos se passarão antes que todo o milhão de bits de dados do usuário ativo seja transmitido. No caso da comutação de pacotes, o usuário ativo poderá enviá-los continuamente à taxa total de 1 Mbit/s, visto que não haverá outros gerando pacotes que precisem ser multiplexados com os dele. Nesse caso, todos os dados do usuário ativo serão transmitidos dentro de 1 segundo.

Os exemplos citados ilustram duas maneiras pelas quais o desempenho da comutação de pacotes pode ser superior à da comutação de circuitos. Também destacam a diferença crucial entre as duas formas de compartilhar a taxa de transmissão de um enlace entre vários fluxos de bits. A comutação de circuitos aloca previamente a utilização do enlace de transmissão independentemente da demanda, com desperdício de tempo de enlace desnecessário alocado e não utilizado. A comutação de pacotes, por outro lado, aloca utilização de enlace *por demanda.* A capacidade de transmissão do enlace será compartilhada pacote por pacote somente entre usuários que tenham pacotes que precisam ser transmitidos pelo enlace.

Embora tanto a comutação de pacotes quanto a de circuitos predominem nas redes de telecomunicação de hoje, a tendência é, sem dúvida, a comutação de pacotes. Até mesmo muitas das atuais redes de telefonia de comutação de circuitos estão migrando aos poucos para a comutação de pacotes. Em especial, redes telefônicas usam comutação de pacotes na parte cara de uma chamada telefônica para o exterior.

1.3.3 Uma rede de redes

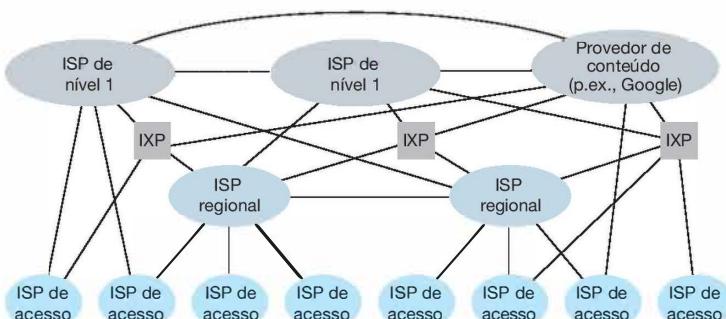
Vimos anteriormente que sistemas finais (PCs, smartphones, servidores Web, servidores de correio eletrônico e assim por diante) conectam-se à Internet por meio de um provedor local (ISP). Este pode fornecer uma conectividade tanto com ou sem fio, utilizando diversas tecnologias de acesso, que incluem DSL, cabo, FTTH, Wi-Fi e telefone celular. Observe que o provedor local não precisa ser uma operadora de telefonia ou uma empresa de TV a cabo: pode ser, por exemplo, uma universidade (que oferece acesso à Internet para os alunos, os funcionários e o corpo docente) ou uma empresa (que oferece acesso para seus funcionários). Mas conectar usuários finais e provedores de conteúdo a um provedor de acesso (ISP) é apenas uma pequena peça do quebra-cabeça

que é interligar os bilhões de sistemas finais que compõem a Internet. Isso é feito criando uma *rede de redes* — entender essa frase é a chave para entender a Internet.

Com o passar dos anos, a rede de redes que forma a Internet evoluiu para uma estrutura bastante complexa. Grande parte dessa evolução é controlada pela política econômica e nacional, e não por considerações de desempenho. Para entender a estrutura de rede da Internet de hoje, vamos criar, de modo incremental, uma série de estruturas de rede, com cada nova estrutura sendo uma aproximação melhor da Internet complexa que temos. Lembre-se de que o objetivo dominante é interconectar os provedores de acesso de modo que todos os sistemas finais possam enviar pacotes entre si. Um método ingênuo seria fazer que cada ISP se conectasse *diretamente* a cada outro ISP. Esse projeto em malha, é evidente, seria muito caro para os ISPs, pois exigiria que cada ISP tivesse um enlace de comunicação separado para as centenas de milhares de outros ISPs do mundo inteiro.

Nossa primeira estrutura de rede, a *Estrutura de Rede 1*, interconecta todos os ISPs de acesso a um único *ISP de trânsito global*. Nosso (imaginário) ISP de trânsito global é uma rede de roteadores e enlaces de comunicação que não apenas se espalha pelo planeta, mas também tem pelo menos um roteador próximo de cada uma das centenas de milhares de ISPs de acesso. Claro, seria muito dispendioso para o ISP global montar essa rede tão extensa. Para que seja lucrativo, ele naturalmente cobraria de cada um dos ISPs de acesso pela conectividade, com o preço refletindo (mas nem sempre diretamente proporcional à) a quantidade de tráfego que um ISP de acesso troca com o ISP global. Como o ISP de acesso paga ao ISP de trânsito global, ele é considerado um **cliente**, e o ISP de trânsito global é considerado um **provedor**.

Agora, se alguma empresa montar e operar um ISP de trânsito global que seja lucrativo, então será natural para outras empresas montarem seus próprios ISPs de trânsito global e competirem com o original. Isso leva à *Estrutura de Rede 2*, que consiste em centenas de milhares de ISPs de acesso e *múltiplos* ISPs de trânsito global. Os ISPs de acesso decerto preferem a Estrutura de Rede 2 à Estrutura de Rede 1, pois agora podem escolher entre os provedores de trânsito global concorrentes comparando seus preços e serviços. Note, porém, que os próprios ISPs de trânsito global precisam se interconectar: caso contrário, os ISPs de acesso conectados a um dos provedores de trânsito global não poderiam se comunicar com os ISPs de acesso conectados aos outros provedores de trânsito global.

A Estrutura de Rede 2, que acabamos de descrever, é uma hierarquia de duas camadas com provedores de trânsito global residindo no nível superior e os ISPs de acesso no nível inferior. Isso considera que os ISPs de trânsito global não são capazes de chegar perto de todo e qualquer ISP de acesso, mas também consideram economicamente desejável fazer isso. Na realidade, embora alguns ISPs tenham uma cobertura global impressionante e se conectem diretamente com muitos ISPs de acesso, nenhum tem presença em toda e qualquer cidade do mundo. Em vez disso, em determinada região, pode haver um **ISP regional** ao qual os ISPs de acesso na região se conectam. Cada ISP regional, então, se conecta a **ISPs de nível 1**. Estes são semelhantes ao nosso (imaginário) ISP de trânsito global; mas os ISPs de nível 1, que realmente existem, não têm uma presença em cada cidade do mundo. Existe mais ou menos uma dúzia de ISPs de nível 1, incluindo Level 3 Communications, AT&T, Sprint e NTT. É interessante que nenhum grupo sanciona oficialmente o *status* de nível 1; como diz o ditado — se você tiver que perguntar se é membro de um grupo, provavelmente não é.

Retornando a essa rede de redes, não apenas existem vários ISPs de nível 1 concorrentes, mas pode haver múltiplos ISPs regionais concorrentes em uma região. Em tal hierarquia, cada ISP de acesso paga ao regional ao qual se conecta, e cada ISP regional paga ao ISP de nível 1 ao qual se interliga. (Um ISP de acesso também pode se conectar diretamente a um ISP de nível 1, quando pagará ao ISP de nível 1.) Assim, existe uma relação cliente -provedor em cada nível da hierarquia. Observe que os ISPs de nível 1 não pagam a ninguém, pois estão no topo. Para complicar as coisas ainda mais, em algumas regiões pode haver um ISP regional maior (talvez se espalhando por um país inteiro) ao qual os ISPs regionais menores nessa região se conectam; o ISP regional maior, então, se conecta a um ISP de nível 1. Por exemplo, na China existem ISPs de acesso em cada cidade, que se conectam a ISPs provinciais, que por sua vez se ligam a ISPs nacionais, que por fim se interligam a ISPs de nível 1 [Tian, 2012]. Chamamos a essa hierarquia multinível, que ainda é apenas uma aproximação bruta da Internet de hoje, *Estrutura de Rede 3*.

Para montar uma rede que se assemelhe mais à Internet de hoje, temos que acrescentar pontos de presença (PoPs — Points of Presence), *multi-homing*, emparelhamento e pontos de troca da Internet (IXPs — Internet eXchange Points) à Estrutura de Rede 3. Existem PoPs em todos os níveis da hierarquia, exceto para o nível de baixo (ISP de acesso). Um **PoP** é simplesmente um grupo de um ou mais roteadores (no mesmo local) na rede do provedor, onde os ISPs clientes podem se conectar no ISP provedor. Para que uma rede do cliente se conecte ao PoP de um provedor, ele pode alugar um enlace de alta velocidade de um provedor de telecomunicações de terceiros para conectar diretamente um de seus roteadores a um roteador no PoP. Qualquer ISP (exceto os de nível 1) pode decidir efetuar o ***multi-home***, ou seja, conectar-se a dois ou mais ISPs provedores. Assim, por exemplo, um ISP de acesso pode efetuar *multi-home* com dois ISPs regionais, ou então com dois ISPs regionais e também com um ISP de nível 1. De modo semelhante, um ISP regional pode efetuar *multi-home* com vários ISPs de nível 1. Quando um ISP efetua *multi-home*, ele pode continuar a enviar e receber pacotes na Internet, mesmo que um de seus provedores apresente uma falha.

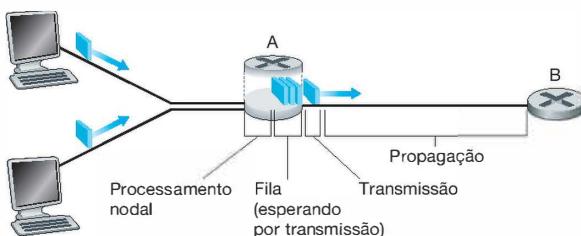
Como vimos, os ISPs clientes pagam aos seus ISPs provedores para obter interconectividade global com a Internet. O valor que um ISP cliente paga a um ISP provedor reflete a quantidade de tráfego que ele troca com o provedor. Para reduzir esses custos, um par de ISPs próximos no mesmo nível da hierarquia pode **emparelhar**, ou seja, conectar diretamente suas redes, de modo que todo o tráfego entre elas passe pela conexão direta, em vez de passar por intermediários mais à frente. Quando dois ISPs são emparelhados, isso em geral é feito em acordo, ou seja, nenhum ISP paga ao outro. Como já dissemos, os ISPs de nível 1 também são emparelhados uns com os outros, sem taxas. Para ver uma discussão legível sobre emparelhamento e relações cliente-provedor, consulte Van der Berg [2008]. Nesses mesmos termos, uma empresa de terceiros pode criar um **ponto de troca da Internet (IXP — Internet Exchange Point)** — quase sempre em um prédio isolado com seus próprios comutadores —, que é um ponto de encontro onde vários ISPs podem se emparelhar. Existem cerca de 300 IXPs na Internet hoje [Augustin, 2009]. Referimo-nos a esse ecossistema — consistindo em ISPs de acesso, ISPs regionais, ISPs de nível 1, PoPs, *multi-homing*, emparelhamento e IXPs — como *Estrutura de Rede 4*.

Agora, chegamos finalmente na *Estrutura de Rede 5*, que descreve a Internet de 2012. Essa estrutura, ilustrada na Figura 1.15, se baseia no topo da Estrutura de Rede 4 acrescentando **redes de provedor de conteúdo**. A Google é um dos principais exemplos dessa rede de provedor de conteúdo. No momento, estima-se que tenha de 30 a 50 centros de dados distribuídos na América do Norte, Europa, Ásia, América do Sul e Austrália. Alguns desses centros de dados acomodam mais de cem mil servidores, enquanto outros são menores, acomodando apenas centenas de servidores. Os centros de dados da Google são todos interconectados por meio de uma rede TCP/IP privativa, que se espalha pelo mundo inteiro, mas apesar disso é separada da Internet pública. O importante é que essa rede privada só transporta tráfego de/para servidores da Google. Como vemos na Figura 1.15, a rede privativa da Google tenta "contornar" as camadas mais altas da Internet emparelhando (sem custo) com outros ISPs de nível mais baixo, seja conectando diretamente ou interligando com eles em IXPs [Labovitz, 2010]. Entretanto, como muitos ISPs de acesso ainda

FIGURA 1.15 INTERCONEXÃO DE ISPs

só podem ser alcançados transitando por redes de nível 1, a rede da Google também se conecta a ISPs de nível 1 e paga a esses ISPs pelo tráfego que troca com eles. Criando sua própria rede, um provedor de conteúdo não apenas reduz seus pagamentos aos ISPs da camada mais alta, mas também tem maior controle de como seus serviços por fim são entregues aos usuários finais. A infraestrutura de rede da Google é descrita com mais detalhes na Seção 7.2.4.

Resumindo, a topologia da Internet é complexa, consistindo em uma dúzia ou mais de ISPs de nível 1 e centenas de milhares de ISPs de níveis mais baixos. A cobertura dos ISPs é bastante diversificada; alguns abrangem vários continentes e oceanos e outros se limitam a pequenas regiões geográficas. Os ISPs de níveis mais baixos conectam-se a ISPs de níveis mais altos e estes se interconectam uns com os outros. Usuários e provedores de conteúdo são clientes de ISPs de níveis mais baixos e estes são clientes de ISPs de níveis mais altos. Nos últimos anos, os principais provedores de conteúdo também têm criado suas próprias redes e se conectam diretamente a ISPs de níveis mais baixos, quando possível.

1.4 ATRASO, PERDA E VAZÃO EM REDES DE COMUTAÇÃO DE PACOTES

Na Seção 1.1 dissemos que a Internet pode ser vista como uma infraestrutura que fornece serviços a aplicações distribuídas que são executadas nos sistemas finais. De modo ideal, gostaríamos que os serviços da Internet transferissem tantos dados quanto desejamos entre dois sistemas finais, de modo instantâneo, sem nenhuma perda. É uma pena, mas esse é um objetivo muito elevado, algo inalcançável. Em vez disso, as redes de computadores, necessariamente, restringem a vazão (a quantidade de dados por segundo que podem ser transferidos) entre sistemas finais, apresentam atrasos entre sistemas finais e podem perder pacotes. Por um lado, infelizmente as leis físicas da realidade introduzem atraso e perda, bem como restringem a vazão. Por outro, como as redes de computadores têm esses problemas, existem muitas questões fascinantes sobre como lidar com eles — questões mais do que suficientes para preencher um curso de redes de computadores e motivar milhares de teses de doutorado! Nesta seção, começaremos a examinar e quantificar atraso, perda e vazão em redes de computadores.

1.4.1 Uma visão geral de atraso em redes de comutação de pacotes

Lembre-se de que um pacote começa em um sistema final (a origem), passa por uma série de roteadores e termina sua jornada em outro sistema final (o destino). Quando um pacote viaja de um nó (sistema final ou roteador) ao nó subsequente (sistema final ou roteador), sofre, ao longo desse caminho, diversos tipos de atraso em *cada* nó. Os mais importantes deles são o **atraso de processamento nodal**, o **atraso de fila**, o **atraso de transmissão** e o **atraso de propagação**; juntos, eles se acumulam para formar o **atraso nodal total**. O desempenho de muitas aplicações da Internet — como busca, navegação Web, e-mail, mapas, mensagens instantâneas e voz sobre IP — é bastante afetado por atrasos na rede. Para entender a fundo a comutação de pacotes e redes de computadores, é preciso entender a natureza e a importância desses atrasos.

Tipos de atraso

Vamos examinar esses atrasos no contexto da Figura 1.16. Como parte de sua rota fim a fim entre origem e destino, um pacote é enviado do nó anterior por meio do roteador A até o roteador B. Nossa meta é caracterizar o atraso nodal no roteador A. Note que este tem um enlace de saída que leva ao roteador B. Esse enlace é precedido de uma fila (também conhecida como buffer). Quando o pacote chega ao roteador A, vindo do nó anterior, o roteador examina o cabeçalho do pacote para determinar o enlace de saída apropriado e então o direciona ao enlace. Nesse exemplo, o enlace de saída para o pacote é o que leva ao roteador B. Um pacote pode ser transmitido por um enlace apenas se não houver nenhum outro sendo transmitido por ele e se não houver outros à sua frente na fila. Se o enlace estiver ocupado, ou com pacotes à espera, o recém-chegado entrará na fila.

FIGURA 1.16 O ATRASO NODAL NO ROTEADOR A

Atraso de processamento

O tempo exigido para examinar o cabeçalho do pacote e determinar para onde direcioná-lo é parte do **atraso de processamento**, que pode também incluir outros fatores, como o tempo necessário para verificar os erros em bits existentes no pacote que ocorreram durante a transmissão dos bits desde o nó anterior ao roteador A. Atrasos de processamento em roteadores de alta velocidade em geral são da ordem de microssegundos, ou menos. Depois desse processamento nodal, o roteador direciona o pacote à fila que precede o enlace com o roteador B. (No Capítulo 4, estudaremos os detalhes da operação de um roteador.)

Atraso de fila

O pacote sofre um **atraso de fila** enquanto espera para ser transmitido no enlace. O tamanho desse atraso dependerá da quantidade de outros pacotes que chegarem antes e que já estiverem na fila esperando pela transmissão no enlace. Se a fila estiver vazia, e nenhum outro pacote estiver sendo transmitido naquele momento, então o tempo de fila de nosso pacote será zero. Por outro lado, se o tráfego estiver intenso e houver muitos pacotes também esperando para ser transmitidos, o atraso de fila será longo. Em breve, veremos que o número de pacotes que um determinado pacote provavelmente encontrará ao chegar é uma função da intensidade e da natureza do tráfego que está chegando à fila. Na prática, atrasos de fila podem ser da ordem de micro a milissegundos.

Atraso de transmissão

Admitindo-se que pacotes são transmitidos segundo a estratégia de "o primeiro a chegar será o primeiro a ser processado", como é comum em redes de comutação de pacotes, o nosso somente poderá ser transmitido depois de todos os que chegaram antes terem sido enviados. Denominemos o tamanho do pacote como L bits e a velocidade de transmissão do enlace do roteador A ao roteador B como R bits/s. Por exemplo, para um enlace Ethernet de 10 Mbits/s, a velocidade é R = 10 Mbits/s; para um enlace Ethernet de 100 Mbits/s, a velocidade é R = 100 Mbits/s. O **atraso de transmissão** é L/R. Esta é a quantidade de tempo exigida para empurrar (isto é, transmitir) todos os bits do pacote para o enlace. Na prática, atrasos de transmissão costumam ser da ordem de micro a milissegundos.

Atraso de propagação

Assim que é lançado no enlace, um bit precisa se propagar até o roteador B. O tempo necessário para propagar o bit desde o início do enlace até o roteador B é o **atraso de propagação**. O bit se propaga à velocidade de propagação do enlace, a qual depende do meio físico (isto é, fibra ótica, par de fios de cobre trançado e assim por diante) e está na faixa de

$$2 \cdot 10^8 \text{ m/s a } 3 \cdot 10^8 \text{ m/s}$$

que é igual à velocidade da luz. O atraso de propagação é a distância entre dois roteadores dividida pela velocidade de propagação. Isto é, o atraso de propagação é d/s, sendo d a distância entre o roteador A e o roteador B, e s a velocidade de propagação do enlace. Assim que o último bit do pacote se propagar até o nó B, ele e todos os outros bits precedentes serão armazenados no roteador B. Então, o processo inteiro continua, agora com o roteador B executando a retransmissão. Em redes WAN, os atrasos de propagação são da ordem de milissegundos.

Comparação entre atrasos de transmissão e de propagação

Os principiantes na área de redes de computadores às vezes têm dificuldade para entender a diferença entre atrasos de transmissão e de propagação. Ela é sutil, mas importante. O atraso de transmissão é a quantidade de tempo necessária para o roteador empurrar o pacote para fora; é uma função do comprimento do pacote e da taxa de transmissão do enlace, mas nada tem a ver com a distância entre os roteadores. O atraso de propagação, por outro lado, é o tempo que leva para um bit se propagar de um roteador até o seguinte; é uma função da distância entre os roteadores, mas nada tem a ver com o comprimento do pacote ou com a taxa de transmissão do enlace.

Podemos esclarecer melhor as noções de atrasos de transmissão e de propagação com uma analogia. Considere uma rodovia que tenha um posto de pedágio a cada 100 quilômetros, como mostrado na Figura 1.17. Imagine que os trechos da rodovia entre os postos de pedágio sejam enlaces e que os postos de pedágio sejam roteadores. Suponha que os carros trafeguem (isto é, se propaguem) pela rodovia a uma velocidade de 100 km/h (isto é, quando o carro sai de um posto de pedágio, acelera instantaneamente até 100 km/h e mantém essa velocidade entre os dois postos de pedágio). Agora, considere que dez carros viajem em comboio, um atrás do outro, em ordem fixa. Imagine que cada carro seja um bit e que o comboio seja um pacote. Suponha ainda que cada posto de pedágio libere (isto é, transmita) um carro a cada 12 segundos, que seja tarde da noite e que os carros do comboio sejam os únicos na estrada. Por fim, imagine que, ao chegar a um posto de pedágio, o primeiro carro do comboio aguarde na entrada até que os outros nove cheguem e formem uma fila atrás dele. (Assim, o comboio inteiro deve ser "armazenado" no posto de pedágio antes de começar a ser "reenviado".) O tempo necessário para que todo o comboio passe pelo posto de pedágio e volte à estrada é de (10 carros)/(5 carros/minuto) = 2 minutos, semelhante ao atraso de transmissão em um roteador. O tempo necessário para um carro trafegar da saída de um posto de pedágio até o próximo é de (100 km)/(100 km/h) = 1 hora, semelhante ao atraso de propagação. Portanto, o tempo decorrido entre o instante em que o comboio é "armazenado" em frente a um posto de pedágio até o momento em que é "armazenado" em frente ao seguinte é a soma do atraso de transmissão e do atraso de propagação — nesse exemplo, 62 minutos.

Vamos explorar um pouco mais essa analogia. O que aconteceria se o tempo de liberação do comboio no posto de pedágio fosse maior do que o tempo que um carro leva para trafegar entre dois postos? Por exemplo, suponha que os carros trafeguem a uma velocidade de 1.000 km/h e que o pedágio libere um carro por minuto. Então, o atraso de trânsito entre dois postos de pedágio é de 6 minutos e o tempo de liberação do comboio no posto de pedágio é de 10 minutos. Nesse caso, os primeiros carros do comboio chegarão ao segundo posto de pedágio antes que os últimos carros saiam do primeiro posto. Essa situação também acontece em redes de comutação de pacotes — os primeiros bits de um pacote podem chegar a um roteador enquanto muitos dos remanescentes ainda estão esperando para ser transmitidos pelo roteador precedente.

FIGURA 1.17 ANALOGIA DO COMBOIO

Se uma imagem vale mil palavras, então uma animação vale um milhão. O site de apoio deste livro apresenta um aplicativo interativo Java que ilustra e compara o atraso de transmissão com o de propagação. Recomenda-se que o leitor visite esse aplicativo. Smith [2009] também oferece uma discussão bastante legível sobre atrasos de propagação, fila e transmissão.

Se d_{proc}, d_{fila}, d_{trans} e d_{prop} forem, respectivamente, os atrasos de processamento, de fila, de transmissão e de propagação, então o atraso nodal total é dado por:

$$d_{nodal} = d_{proc} + d_{fila} + d_{trans} + d_{prop}$$

A contribuição desses componentes do atraso pode variar significativamente. Por exemplo, d_{prop} pode ser desprezível (por exemplo, dois microssegundos) para um enlace que conecta dois roteadores no mesmo *campus* universitário; contudo, é de centenas de milissegundos para dois roteadores interconectados por um enlace de satélite geoestacionário e pode ser o termo dominante no d_{nodal}. De maneira semelhante, d_{trans} pode variar de desprezível a significativo. Sua contribuição costuma ser desprezível para velocidades de transmissão de 10 Mbits/s e mais altas (por exemplo, em LANs); contudo, pode ser de centenas de milissegundos para grandes pacotes de Internet enviados por enlaces de modems discados de baixa velocidade. O atraso de processamento, d_{proc}, é quase sempre desprezível; no entanto, tem forte influência sobre a produtividade máxima de um roteador, que é a velocidade máxima com que ele pode encaminhar pacotes.

1.4.2 Atraso de fila e perda de pacote

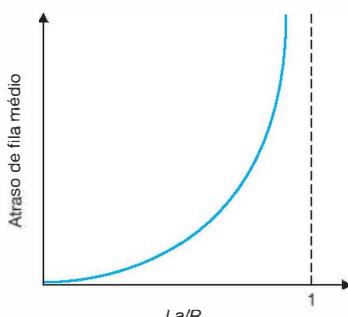
O mais complicado e interessante componente do atraso nodal é o atraso de fila, d_{fila}. Na verdade, o atraso de fila é tão importante e interessante em redes de computadores que milhares de artigos e numerosos livros já foram escritos sobre ele [Bertsekas, 1991; Daigle, 1991; Kleinrock, 1975, 1976; Ross, 1995]. Neste livro, faremos apenas uma discussão intuitiva, de alto nível, sobre o atraso de fila; o leitor mais curioso pode consultar alguns dos livros citados (ou até mesmo escrever uma tese sobre o assunto!). Diferente dos três outros atrasos (a saber, d_{proc}, d_{trans} e d_{prop}), o atraso de fila pode variar de pacote a pacote. Por exemplo, se dez pacotes chegarem a uma fila vazia ao mesmo tempo, o primeiro pacote transmitido não sofrerá nenhum atraso de fila, ao passo que o último sofrerá um relativamente grande (enquanto espera que os outros nove sejam transmitidos). Por conseguinte, para se caracterizar um atraso de fila, são utilizadas em geral medições estatísticas, tais como atraso de fila médio, variância do atraso de fila e a probabilidade de que ele exceda um valor especificado.

Quando o atraso de fila é grande e quando é insignificante? A resposta a essa pergunta depende da velocidade de transmissão do enlace, da taxa com que o tráfego chega à fila e de sua natureza, isto é, se de modo intermitente, em rajadas. Para entendermos melhor, vamos adotar a para representar a taxa média com que os pacotes chegam à fila (a é medida em pacotes/segundo). Lembre-se de que R é a taxa de transmissão, isto é, a taxa (em bits/segundo) com que os bits são retirados da fila. Suponha também, para simplificar, que todos os pacotes tenham L bits. Então, a taxa média com que os bits chegam à fila é La bits/s. Por fim, imagine que a fila seja muito longa, de modo que possa conter um número infinito de bits. A razão La/R, denominada **intensidade de tráfego**, costuma desempenhar um papel importante na estimativa do tamanho do atraso de fila. Se $La/R > 1$, então a velocidade média com que os bits chegam à fila excederá aquela com que eles podem ser transmitidos para fora da fila. Nessa situação desastrosa, a fila tenderá a aumentar sem limite e o atraso de fila tenderá ao infinito! Por conseguinte, uma das regras de ouro da engenharia de tráfego é: *projete seu sistema de modo que a intensidade de tráfego não seja maior do que 1.*

Agora, considere o caso em que $La/R \leq 1$. Aqui, a natureza do tráfego influencia o atraso de fila. Por exemplo, se pacotes chegarem periodicamente — isto é, se chegar um pacote a cada L/R segundos —, então todos os pacotes chegarão a uma fila vazia e não haverá atraso. Por outro lado, se chegarem em rajadas, mas periodicamente, poderá haver um significativo atraso de fila médio. Por exemplo, suponha que N pacotes cheguem ao mesmo tempo a cada $(L/R)N$ segundos. Então, o primeiro pacote transmitido não sofrerá atraso de fila; o segundo terá

um atraso de L/R segundos e, de modo mais geral, o enésimo pacote transmitido terá um atraso de fila de $(n - 1)$ L/R segundos. Deixamos como exercício para o leitor o cálculo do atraso de fila médio para esse exemplo.

Os dois exemplos de chegadas periódicas que acabamos de descrever são um tanto acadêmicos. Em geral, o processo de chegada a uma fila é *aleatório* — isto é, não segue um padrão e os intervalos de tempo entre os pacotes são ao acaso. Nessa hipótese mais realista, a quantidade La/R quase sempre não é suficiente para caracterizar por completo a estatística do atraso. Não obstante, é útil para entender intuitivamente a extensão do atraso de fila. Em especial, se a intensidade de tráfego for próxima de zero, então as chegadas de pacotes serão poucas e bem espaçadas e é improvável que um pacote que esteja chegando encontre outro na fila. Consequentemente, o atraso de fila médio será próximo de zero. Por outro lado, quando a intensidade de tráfego for próxima de 1, haverá intervalos de tempo em que a velocidade de chegada excederá a capacidade de transmissão (por causa das variações na taxa de chegada do pacote) e uma fila será formada durante esses períodos; quando a taxa de chegada for menor do que a capacidade de transmissão, a extensão da fila diminuirá. Todavia, à medida que a intensidade de tráfego se aproxima de 1, o comprimento médio da fila fica cada vez maior. A dependência qualitativa entre o atraso de fila médio e a intensidade de tráfego é mostrada na Figura 1.18.

Um aspecto importante a observar na Figura 1.18 é que, quando a intensidade de tráfego se aproxima de 1, o atraso de fila médio aumenta depressa. Uma pequena porcentagem de aumento na intensidade resulta em um aumento muito maior no atraso, em termos de porcentagem. Talvez você já tenha percebido esse fenômeno na estrada. Se você dirige regularmente por uma estrada que costuma ser congestionada, o fato de ela estar sempre assim significa que a intensidade de tráfego é próxima de 1. Se algum evento causar um tráfego um pouco maior do que o normal, as demoras que você sofrerá poderão ser enormes.

Para compreender um pouco mais os atrasos de fila, visite o site de apoio do livro, que apresenta um aplicativo Java interativo para uma fila. Se você aumentar a taxa de chegada do pacote o suficiente de forma que a intensidade do tráfego exceda 1, verá a fila aumentar ao longo do tempo.

Perda de pacotes

Na discussão anterior, admitimos que a fila é capaz de conter um número infinito de pacotes. Na realidade, a capacidade da fila que precede um enlace é finita, embora a sua formação dependa bastante do projeto e do custo do comutador. Como a capacidade da fila é finita, na verdade os atrasos de pacote não se aproximam do infinito quando a intensidade de tráfego se aproxima de 1. O que acontece de fato é que um pacote pode chegar e encontrar uma fila cheia. Sem espaço disponível para armazená-lo, o roteador o **descartará**; isto é, ele será **perdido**. Esse excesso em uma fila pode ser observado novamente no aplicativo Java quando a intensidade do tráfego é maior do que 1.

Do ponto de vista de um sistema final, uma perda de pacote é vista como um pacote que foi transmitido para o núcleo da rede, mas sem nunca ter emergido dele no destino. A fração de pacotes perdidos aumenta com

FIGURA 1.18 DEPENDÊNCIA ENTRE ATRASO DE FILA MÉDIO E INTENSIDADE DE TRÁFEGO

o aumento da intensidade de tráfego. Por conseguinte, o desempenho em um nó costuma ser medido não apenas em termos de atraso, mas também da probabilidade de perda de pacotes. Como discutiremos nos capítulos subsequentes, um pacote perdido pode ser retransmitido fim a fim para garantir que todos os dados sejam transferidos da origem ao local de destino.

1.4.3 Atraso fim a fim

Até o momento, nossa discussão focalizou o atraso nodal, isto é, em um único roteador. Concluiremos essa discussão considerando brevemente o atraso da origem ao destino. Para entender esse conceito, suponha que haja $N - 1$ roteadores entre a máquina de origem e a de destino. Imagine também que a rede não esteja congestionada (e, portanto, os atrasos de fila sejam desprezíveis), que o atraso de processamento em cada roteador e na máquina de origem seja d_{proc}, que a taxa de transmissão de saída de cada roteador e da máquina de origem seja R bits/s e que atraso de propagação em cada enlace seja d_{prop}. Os atrasos nodais se acumulam e resultam em um atraso fim a fim,

$$d_{\text{fim a fim}} = N\,(d_{proc} + d_{trans} + d_{prop}) \qquad (1.2)$$

em que, mais uma vez, $d_{trans} = L/R$ e L é o tamanho do pacote. Note que a Equação 1.2 é uma generalização da Equação 1.1, na qual não levamos em conta os atrasos de processamento e propagação. Convidamos você a generalizar a Equação 1.2 para o caso de atrasos heterogêneos nos nós e para o caso de um atraso de fila médio em cada nó.

Traceroute

Para perceber o que é de fato o atraso em uma rede de computadores, podemos utilizar o Traceroute, programa de diagnóstico que pode ser executado em qualquer hospedeiro da Internet. Quando o usuário especifica um nome de hospedeiro de destino, o programa no hospedeiro de origem envia vários pacotes especiais em direção àquele destino. Ao seguir seu caminho até o destino, esses pacotes passam por uma série de roteadores. Um deles recebe um desses pacotes especiais e envia à origem uma curta mensagem, contendo o nome e o endereço do roteador.

Mais especificamente, suponha que haja $N - 1$ roteadores entre a origem e o destino. Então, a fonte enviará N pacotes especiais à rede e cada um deles estará endereçado ao destino final. Esses N pacotes especiais serão marcados de *1* a N, sendo a marca do primeiro pacote *1* e a do último, N. Assim que o *enésimo* roteador recebe o *enésimo* pacote com a marca n, não envia o pacote a seu destino, mas uma mensagem à origem. Quando o hospedeiro de destino recebe o pacote N, também envia uma mensagem à origem, que registra o tempo transcorrido entre o envio de um pacote e o recebimento da mensagem de retorno correspondente. A origem registra também o nome e o endereço do roteador (ou do hospedeiro de destino) que retorna a mensagem. Dessa maneira, a origem pode reconstruir a rota tomada pelos pacotes que vão da origem ao destino e pode determinar os atrasos de ida e volta para todos os roteadores intervenientes. Na realidade, o programa Traceroute repete o processo que acabamos de descrever três vezes, de modo que a fonte envia, na verdade, $3 \cdot N$ pacotes ao destino. O RFC 1393 descreve detalhadamente o Traceroute.

Eis um exemplo de resultado do programa Traceroute, no qual a rota traçada ia do hospedeiro de origem gaia.cs.umass.edu (na Universidade de Massachusetts) até cis.poly.edu (na Polytechnic University no Brooklyn). O resultado tem seis colunas: a primeira é o valor n descrito, isto é, o número do roteador ao longo da rota; a segunda é o nome do roteador; a terceira é o endereço do roteador (na forma xxx.xxx.xxx.xxx); as últimas três são os atrasos de ida e volta para três tentativas. Se a fonte receber menos do que três mensagens de qualquer roteador determinado (por causa da perda de pacotes na rede), o Traceroute coloca um asterisco logo após o número do roteador e registra menos do que três tempos de duração de ida e volta para aquele roteador.

```
1 cs-gw (128.119.240.254) 1.009 ms 0.899 ms 0.993 ms
2 128.119.3.154 (128.119.3.154) 0.931 ms 0.441 ms 0.651 ms
3 border4-rt-gi-1-3.gw.umass.edu (128.119.2.194) 1.032 ms 0.484 ms 0.451 ms
4 acr1-ge-2-1-0.Boston.cw.net (208.172.51.129) 10.006 ms 8.150 ms 8.460 ms
5 agr4-loopback.NewYork.cw.net (206.24.194.104) 12.272 ms 14.344 ms 13.267 ms
6 acr2-loopback.NewYork.cw.net (206.24.194.62) 13.225 ms 12.292 ms 12.148 ms
7 pos10-2.core2.NewYork1.Level3.net (209.244.160.133) 12.218 ms 11.823 ms 11.793 ms
8 gige9-1-52.hsipaccess1.NewYork1.Level3.net (64.159.17.39) 13.081 ms 11.556 ms 13.297
  ms
9 p0-0.polyu.bbnplanet.net (4.25.109.122) 12.716 ms 13.052 ms 12.786 ms
10 cis.poly.edu (128.238.32.126) 14.080 ms 13.035 ms 12.802 ms
```

No exemplo anterior há nove roteadores entre a origem e o destino. Quase todos eles têm um nome e todos têm endereços. Por exemplo, o nome do Roteador 3 é `border4-rt-gi-1-3.gw.umass.edu` e seu endereço é `128.119.2.194`. Examinando os dados apresentados para ele, verificamos que, na primeira das três tentativas, o atraso de ida e volta entre a origem e o roteador foi de 1,03 ms. Os atrasos de ida e volta para as duas tentativas subsequentes foram 0,48 e 0,45 ms, e incluem todos os atrasos que acabamos de discutir, ou seja, de transmissão, de propagação, de processamento do roteador e de fila. Como o atraso de fila varia com o tempo, o atraso de ida e volta do pacote *n* enviado a um roteador *n* pode, às vezes, ser maior do que o do pacote *n*+1 enviado ao roteador *n*+1. Realmente, observamos esse fenômeno no exemplo anterior: os atrasos do roteador 6 são maiores que os do roteador 7!

Você quer experimentar o Traceroute por conta própria? Recomendamos *muito* que visite o site <http://www. traceroute.org>, que oferece uma interface Web para uma extensa lista de fontes para traçar rotas. Escolha uma fonte, forneça o nome de hospedeiro para qualquer destino e o programa Traceroute fará todo o trabalho. Existem muitos programas de software gratuitos que apresentam uma interface gráfica para o Traceroute; um dos nossos favoritos é o PingPlotter [PingPlotter, 2012].

Sistema final, aplicativo e outros atrasos

Além dos atrasos de processamento, transmissão e de propagação, os sistemas finais podem adicionar outros atrasos significativos. Por exemplo, um sistema final que quer transmitir um pacote para uma mídia compartilhada (por exemplo, como em um cenário Wi-Fi ou modem a cabo) pode, *intencionalmente*, atrasar sua transmissão como parte de seu protocolo por compartilhar a mídia com outros sistemas finais; vamos analisar tais protocolos em detalhes no Capítulo 5. Outro importante atraso é o atraso de empacotamento de mídia, o qual está presente nos aplicativos VoIP (voz sobre IP). No VoIP, o remetente deve primeiro carregar um pacote com voz digitalizada e codificada antes de transmitir o pacote para a Internet. Esse tempo para carregar um pacote — chamado de atraso de empacotamento — pode ser significativo e ter impacto sobre a qualidade visível pelo usuário de uma chamada VoIP. Esse assunto será explorado mais adiante nos exercícios de fixação no final deste capítulo.

1.4.4 Vazão nas redes de computadores

Além do atraso e da perda de pacotes, outra medida de desempenho importante em redes de computadores é a vazão fim a fim. Para definir vazão, considere a transferência de um arquivo grande do hospedeiro A para o hospedeiro B por uma rede de computadores. Essa transferência pode ser, por exemplo, um videoclipe extenso de um parceiro para outro por meio do sistema de compartilhamento de arquivos P2P. A **vazão instantânea** a qualquer momento é a taxa (em bits/s) em que o hospedeiro B está recebendo o arquivo. (Muitos aplicativos, incluindo muitos sistemas de compartilhamento P2P, exibem a vazão instantânea durante os *downloads* na interface do usuário — talvez você já tenha observado isso!) Se o arquivo consistir em *F* bits e a transferência levar *T* segundos para o hospedeiro B receber todos os *F* bits, então a **vazão média** da transferência do arquivo é *F*/*T* bits/s.

Para algumas aplicações, como a telefonia via Internet, é desejável ter um atraso baixo e uma vazão instantânea acima de algum limiar (por exemplo, superior a 24 kbits/s para aplicações de telefonia via Internet, e superior a 256 kbits/s para algumas aplicações de vídeo em tempo real). Para outras aplicações, incluindo as de transferência de arquivo, o atraso não é importante, mas é recomendado ter a vazão mais alta possível.

Para obter uma visão mais detalhada do importante conceito de vazão, vamos analisar alguns exemplos. A Figura 1.19(a) mostra dois sistemas finais, um servidor e um cliente, conectados por dois enlaces de comunicação e um roteador. Considere a vazão para uma transferência de arquivo do servidor para o cliente. Suponha que R_s seja a taxa do enlace entre o servidor e o roteador; e R_c seja a taxa do enlace entre o roteador e o cliente. Imagine que os únicos bits enviados na rede inteira sejam os do servidor para o cliente. Agora vem a pergunta: nesse cenário ideal, qual é a vazão servidor-para-cliente? Para responder, pense nos bits como um *líquido* e nos enlaces de comunicação como *tubo*. Claro, o servidor não pode enviar os bits através de seu enlace a uma taxa mais rápida do que R_s bits/s; e o roteador não pode encaminhar os bits a uma taxa mais rápida do que R_c bits/s. Se $R_s < R_c$, então os bits enviados pelo servidor "fluirão" diretamente pelo roteador e chegarão ao cliente a uma taxa de R_s bits/s, gerando uma vazão de R_s bits/s. Se, por outro lado, $R_c < R_s$, então o roteador não poderá encaminhar os bits tão rápido quanto ele os recebe. Neste caso, os bits somente deixarão o roteador a uma taxa R_c, dando uma vazão fim a fim de R_c. (Observe também que se os bits continuarem a chegar no roteador a uma taxa R_s, e a deixá-lo a uma taxa R_c, o acúmulo de bits esperando para transmissão ao cliente só aumentará — uma situação, na maioria das vezes, indesejável!) Assim, para essa rede simples de dois enlaces, a vazão é $\text{mín}\{R_c, R_s\}$, ou seja, é a taxa de transmissão do **enlace de gargalo**. Após determinar a vazão, agora podemos aproximar o tempo que leva para transferir um arquivo grande de F bits do servidor ao cliente como $F/\text{mín}\{R_s, R_c\}$. Para um exemplo específico, suponha que você está fazendo o *download* de um arquivo MP3 de F = 32 milhões de bits, o servidor tem uma taxa de transmissão de R_s = 2 Mbits/s, e você tem um enlace de acesso de R_c = 1 Mbit/s. O tempo necessário para transferir o arquivo é, então, 32 segundos. Claro que essas expressões para tempo de vazão e de transferência são apenas aproximações, já que elas não consideram os atrasos para armazenar-e-reenviar e de processamento, bem como assuntos relacionados a protocolos.

A Figura 1.19(b) agora mostra uma rede com N enlaces entre o servidor e o cliente, com as taxas de transmissão R_1, R_2, ..., R_N. Aplicando a mesma análise da rede de dois enlaces, descobrimos que a vazão para uma transferência de arquivo do servidor ao cliente é $\text{mín}\{R_1, R_2, ..., R_N\}$, a qual é novamente a taxa de transmissão do enlace de gargalo ao longo do caminho entre o servidor e o cliente.

Agora considere outro exemplo motivado pela Internet de hoje. A Figura 1.20(a) mostra dois sistemas finais, um servidor e um cliente, conectados a uma rede de computadores. Considere a vazão para uma transferência de arquivo do servidor ao cliente. O servidor está conectado à rede com um enlace de acesso de taxa R_s e o cliente está conectado à rede com um enlace de acesso de R_c. Agora suponha que todos os enlaces no núcleo da rede de comunicação tenham taxas de transmissão muito altas, muito maiores do que R_s e R_c. De fato, hoje, o

FIGURA 1.19 VAZÃO PARA UMA TRANSFERÊNCIA DE ARQUIVO DO SERVIDOR AO CLIENTE

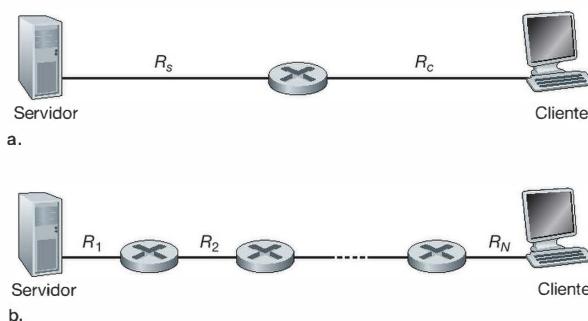
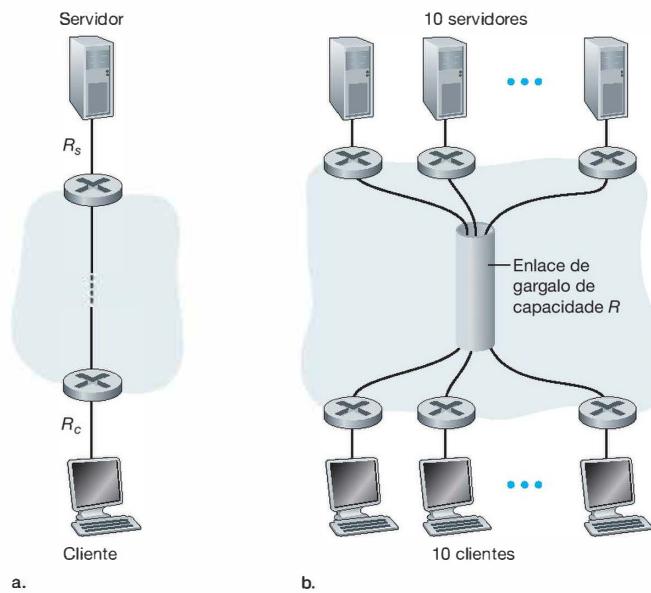



FIGURA 1.20 VAZÃO FIM A FIM: (A) O CLIENTE BAIXA UM ARQUIVO DO SERVIDOR; (B) 10 CLIENTES FAZEM O *DOWNLOAD* COM 10 SERVIDORES

núcleo da Internet está superabastecido com enlaces de alta velocidade que sofrem pouco congestionamento. Suponha, também, que os únicos bits que estão sendo enviados em toda a rede sejam os do servidor para o cliente. Como o núcleo da rede de computadores é como um tubo largo neste exemplo, a taxa a qual os bits correm da origem ao destino é novamente o mínimo de R_s e R_c, ou seja, vazão = mín$\{R_s, R_c\}$. Portanto, o fator restritivo para vazão na Internet de hoje é, em geral, a rede de acesso.

Para um exemplo final, considere a Figura 1.20(b), na qual existem dez servidores e dez clientes conectados ao núcleo da rede de computadores. Nesse exemplo, dez *downloads* simultâneos estão sendo realizados, envolvendo dez pares cliente-servidor. Suponha que esses *downloads* sejam o único tráfego na rede no momento. Como mostrado na figura, há um enlace no núcleo que é atravessado por todos os dez *downloads*. Considere R a taxa de transmissão desse enlace. Imagine que todos os enlaces de acesso do servidor possuem a mesma taxa R_s, todos os enlaces de acesso do cliente possuem a mesma taxa R_c e a taxa de transmissão de todos os enlaces no núcleo — com exceção de um enlace comum de taxa R — sejam muito maiores do que R_s, R_c e R. Agora perguntamos: quais são as vazões de *download*? É claro que se a taxa do enlace comum, R, é grande — digamos, cem vezes maior do que R_s e R_c —, então a vazão para cada *download* será novamente mín$\{R_s, R_c\}$. Mas e se essa taxa for da mesma ordem que R_s e R_c? Qual será a vazão nesse caso? Vamos observar um exemplo específico. Suponha que R_s = 2 Mbits/s, R_c = 1 Mbit/s, R = 5 Mbits/s, e o enlace comum divide sua taxa de transmissão por igual entre 10 *downloads*. Então, o gargalo para cada *download* não se encontra mais na rede de acesso, mas é o enlace compartilhado no núcleo, que somente fornece para cada *download* 500 kbits/s de vazão. Desse modo, a vazão fim a fim é agora reduzida a 500 kbits/s por *download*.

Os exemplos nas Figuras 1.19 e 1.20(a) mostram que a vazão depende das taxas de transmissão dos enlaces sobre as quais os dados fluem. Vimos que quando não há tráfego interveniente, a vazão pode apenas ser aproximada como a taxa de transmissão mínima ao longo do caminho entre a origem e o local de destino. O exemplo na Figura 1.20(b) mostra que, de modo geral, a vazão depende não somente das taxas de transmissão dos enlaces ao longo do caminho, mas também do tráfego interveniente. Em especial, um enlace com uma alta taxa de transmissão pode, apesar disso, ser o enlace de gargalo para uma transferência de arquivo, caso muitos outros

fluxos de dados estejam também passando por aquele enlace. Analisaremos em mais detalhes a vazão em redes de computadores nos exercícios de fixação e nos capítulos subsequentes.

1.5 CAMADAS DE PROTOCOLO E SEUS MODELOS DE SERVIÇO

Até aqui, nossa discussão demonstrou que a Internet é um sistema *extremamente* complicado e que possui muitos componentes: inúmeras aplicações e protocolos, vários tipos de sistemas finais e conexões entre eles, comutadores de pacotes, além de vários tipos de mídia em nível de enlace. Dada essa enorme complexidade, há alguma esperança de organizar a arquitetura de rede ou, ao menos, nossa discussão sobre ela? Felizmente, a resposta a ambas as perguntas é sim.

1.5.1 Arquitetura de camadas

Antes de tentarmos organizar nosso raciocínio sobre a arquitetura da Internet, vamos procurar uma analogia humana. Na verdade, lidamos com sistemas complexos o tempo todo em nosso dia a dia. Imagine se alguém pedisse que você descrevesse, por exemplo, o sistema de uma companhia aérea. Como você encontraria a estrutura para descrever esse sistema complexo que tem agências de emissão de passagens, pessoal para embarcar a bagagem, para ficar no portão de embarque, pilotos, aviões, controle de tráfego aéreo e um sistema mundial de roteamento de aeronaves? Um modo poderia ser apresentar a relação de uma série de ações que você realiza (ou que outros executam para você) quando voa por uma empresa aérea. Você compra a passagem, despacha suas malas, dirige-se ao portão de embarque e, por fim, entra no avião, que decola e segue uma rota até seu destino. Após a aterrissagem, você desembarca no portão designado e recupera suas malas. Se a viagem foi ruim, você reclama na agência que lhe vendeu a passagem (esforço em vão). Esse cenário é ilustrado na Figura 1.21.

Já podemos notar aqui algumas analogias com redes de computadores: você está sendo despachado da origem ao destino pela companhia aérea; um pacote é despachado da máquina de origem à máquina de destino na Internet. Mas essa não é exatamente a analogia que buscamos. Estamos tentando encontrar alguma *estrutura* na Figura 1.21. Observando-a, notamos que há uma função referente à passagem em cada ponta; há também uma função de bagagem para passageiros que já apresentaram o bilhete e uma de portão de embarque para os que já apresentaram o tíquete e despacharam as malas. Para passageiros que já passaram pelo portão de embarque (isto é, aqueles que já apresentaram a passagem, despacharam a bagagem e passaram pelo portão), há uma função de decolagem e de aterrissagem e, durante o voo, uma função de roteamento do avião. Isso sugere que podemos examinar a funcionalidade na Figura 1.21 na *horizontal*, como mostra a Figura 1.22.

FIGURA 1.21 UMA VIAGEM DE AVIÃO: AÇÕES

Passagem (comprar)
Bagagem (despachar)
Portões (embarcar)
Decolagem
Roteamento da aeronave
Passagem (reclamar)
Bagagem (recuperar)
Portões (desembarcar)
Aterrissagem
Roteamento da aeronave
Roteamento da aeronave

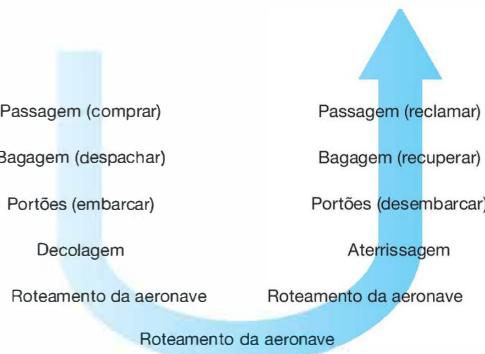
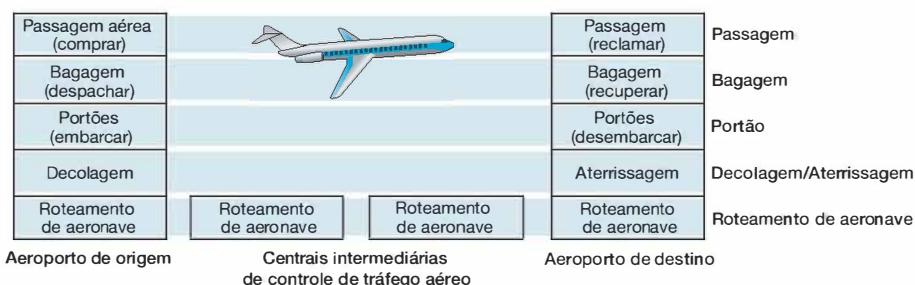



FIGURA 1.22 CAMADAS HORIZONTAIS DA FUNCIONALIDADE DE LINHA AÉREA

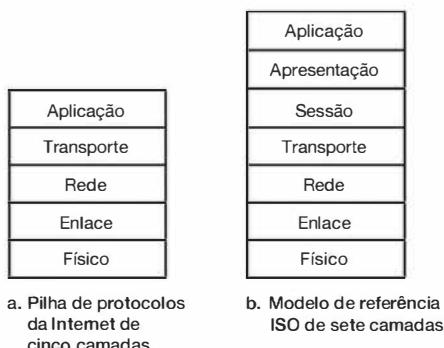
Aeroporto de origem	Centrais intermediárias de controle de tráfego aéreo		Aeroporto de destino	
Passagem aérea (comprar)			Passagem (reclamar)	Passagem
Bagagem (despachar)			Bagagem (recuperar)	Bagagem
Portões (embarcar)			Portões (desembarcar)	Portão
Decolagem			Aterrissagem	Decolagem/Aterrissagem
Roteamento de aeronave	Roteamento de aeronave	Roteamento de aeronave	Roteamento de aeronave	Roteamento de aeronave

A Figura 1.22 dividiu a funcionalidade da linha aérea em camadas, provendo uma estrutura com a qual podemos discutir a viagem aérea. Note que cada camada, combinada com as que estão abaixo dela, implementa alguma funcionalidade, algum *serviço*. Na camada da passagem aérea e abaixo dela, é realizada a transferência "balcão-de-linha-aérea-balcão-de-linha-aérea" de um passageiro. Na camada de bagagem e abaixo dela, é realizada a transferência "despacho-de-bagagem–recuperação-de-bagagem" de um passageiro e de suas malas. Note que a camada da bagagem provê esse serviço apenas para a pessoa que já apresentou o bilhete. Na camada do portão, é realizada a transferência "portão-de-embarque-portão-de-desembarque" do viajante e de suas malas. Na camada de decolagem/aterrissagem, é realizada a transferência "pista-a-pista" de passageiros e de suas bagagens. Cada camada provê seu serviço (1) realizando certas ações dentro dela (por exemplo, na camada do portão, embarcar e desembarcar pessoas de um avião) e (2) utilizando os serviços da camada imediatamente inferior (por exemplo, na do portão, aproveitando o serviço de transferência "pista-a-pista" de passageiros da camada de decolagem/aterrissagem).

Uma arquitetura de camadas nos permite discutir uma parcela específica e bem definida de um sistema grande e complexo. Essa simplificação tem considerável valor intrínseco, pois provê modularidade, tornando muito mais fácil modificar a execução do serviço prestado pela camada. Contanto que a camada forneça o mesmo serviço para a que está acima e use os mesmos serviços da que vem abaixo dela, o restante do sistema permanece inalterado quando a sua realização é modificada. (Note que modificar a implementação de um serviço é muito diferente de mudar o serviço em si!) Por exemplo, se as funções de portão fossem modificadas (digamos que passassem a embarcar e desembarcar passageiros por ordem de altura), o restante do sistema da linha aérea permaneceria inalterado, já que a camada do portão continuaria a prover a mesma função (embarcar e desembarcar passageiros); ela apenas executaria aquela função de maneira diferente após a alteração. Para sistemas grandes e complexos que são atualizados constantemente, a capacidade de modificar a realização de um serviço sem afetar outros componentes do sistema é outra vantagem importante da divisão em camadas.

Camadas de protocolo

Mas chega de linhas aéreas! Vamos agora voltar nossa atenção a protocolos de rede. Para prover uma estrutura para o projeto, projetistas de rede organizam protocolos — e o hardware e o software de rede que os executam — em **camadas**. Cada protocolo pertence a uma das camadas, assim como cada função na arquitetura de linha aérea da Figura 1.22 pertencia a uma camada. Mais uma vez estamos interessados nos **serviços** que uma camada oferece à camada acima dela — denominado **modelo de serviço**. Assim como em nosso exemplo da linha aérea, cada camada provê seu serviço (1) executando certas ações dentro dela e (2) utilizando os serviços da camada diretamente abaixo dela. Por exemplo, os serviços providos pela camada n podem incluir entrega confiável de mensagens de uma extremidade da rede à outra, que pode ser implementada utilizando um serviço não confiável de entrega de mensagem fim a fim da camada $n - 1$ e adicionando funcionalidade da camada n para detectar e retransmitir mensagens perdidas.

Uma camada de protocolo pode ser executada em software, em hardware, ou em uma combinação dos dois. Protocolos de camada de aplicação — como HTTP e SMTP — quase sempre são realizados em software nos sistemas finais; o mesmo acontece com protocolos de camada de transporte. Como a camada física e as de enlace de dados são responsáveis pelo manuseio da comunicação por um enlace específico, em geral são executadas em uma placa de interface de rede (por exemplo, placas de interface Ethernet ou Wi-Fi) associadas a determinado enlace. A camada de rede quase sempre é uma execução mista de hardware e software. Note também que, tal como as funções na arquitetura em camadas da linha aérea eram distribuídas entre os vários aeroportos e centrais de controle de tráfego aéreo que compunham o sistema, um protocolo de camada *n* é *distribuído* entre sistemas finais, comutadores de pacote e outros componentes que formam a rede. Isto é, há sempre uma parte de um protocolo de camada *n* em cada componente de rede.

O sistema de camadas de protocolos tem vantagens conceituais e estruturais [RFC 3439]. Como vimos, a divisão em camadas proporciona um modo estruturado de discutir componentes de sistemas. A modularidade facilita a atualização de componentes de sistema. Devemos mencionar, no entanto, que alguns pesquisadores e engenheiros de rede se opõem veementemente ao sistema de camadas [Wakeman, 1992]. Uma desvantagem potencial é que uma camada pode duplicar a funcionalidade de uma inferior. Por exemplo, muitas pilhas de protocolos oferecem serviço de recuperação de erros para cada enlace e também de fim a fim. Uma segunda desvantagem é que a funcionalidade em uma camada pode necessitar de informações (por exemplo, um valor de carimbo de tempo) que estão presentes somente em outra, o que infringe o objetivo de separação de camadas.

Quando tomados em conjunto, os protocolos das várias camadas são denominados **pilha de protocolos**. A pilha de protocolos da Internet é formada por cinco camadas: física, de enlace, de rede, de transporte e de aplicação, como mostra a Figura 1.23(a). Se você verificar o sumário, verá que organizamos este livro utilizando as camadas da pilha de protocolos da Internet. Fazemos uma **abordagem *top-down*** (de cima para baixo), primeiro abordando a camada de aplicação e prosseguindo para baixo.

Camada de aplicação

A camada de aplicação é onde residem aplicações de rede e seus protocolos. A camada de aplicação da Internet inclui muitos protocolos, tais como o HTTP (que provê requisição e transferência de documentos pela Web), o SMTP (que provê transferência de mensagens de correio eletrônico) e o FTP (que provê a transferência de arquivos entre dois sistemas finais). Veremos que certas funções de rede, como a tradução de nomes fáceis de entender, que são dados a sistemas finais da Internet (por exemplo, de <www.ietf.org> para um endereço de rede de 32 bits), também são executadas com a ajuda de um protocolo de camada de aplicação, no caso, o sistema de nomes de domínio

FIGURA 1.23 A PILHA DE PROTOCOLOS DA INTERNET (A) E O MODELO DE REFERÊNCIA OSI (B)

a. Pilha de protocolos da Internet de cinco camadas

b. Modelo de referência ISO de sete camadas

(*domain name system* — DNS). Veremos no Capítulo 2 que é muito fácil criar nossos próprios novos protocolos de camada de aplicação.

Um protocolo de camada de aplicação é distribuído por diversos sistemas finais, e a aplicação em um sistema final utiliza o protocolo para trocar pacotes de informação com a aplicação em outro sistema final. Chamaremos de **mensagem** a esse pacote de informação na camada de aplicação.

Camada de transporte

A camada de transporte da Internet carrega mensagens da camada de aplicação entre os lados do cliente e servidor de uma aplicação. Há dois protocolos de transporte na Internet: TCP e UDP, e qualquer um pode levar mensagens da camada de aplicação. O TCP provê serviços orientados a conexão para suas aplicações. Alguns desses serviços são a entrega garantida de mensagens da camada de aplicação ao destino e controle de fluxo (isto é, compatibilização das velocidades do remetente e do receptor). O TCP também fragmenta mensagens longas em segmentos mais curtos e provê mecanismo de controle de congestionamento, de modo que uma origem reduz sua velocidade de transmissão quando a rede está congestionada. O protocolo UDP provê serviço não orientado a conexão para suas aplicações. É um serviço econômico que fornece segurança, sem controle de fluxo e de congestionamento. Neste livro, chamaremos de **segmento** a um pacote da camada de transporte.

Camada de rede

A camada de rede da Internet é responsável pela movimentação, de um hospedeiro para outro, de pacotes da camada de rede, conhecidos como **datagramas**. O protocolo de camada de transporte da Internet (TCP ou UDP) em um hospedeiro de origem passa um segmento da camada de transporte e um endereço de destino à camada de rede, exatamente como você passaria ao serviço de correios uma carta com um endereço de destinatário. A camada de rede então provê o serviço de entrega do segmento à camada de transporte no hospedeiro de destino.

Essa camada inclui o famoso protocolo IP, que define os campos no datagrama e o modo como os sistemas finais e os roteadores agem nesses campos. Existe apenas um único protocolo IP, e todos os componentes da Internet que têm uma camada de rede devem executá-lo. A camada de rede da Internet também contém protocolos de roteamento que determinam as rotas que os datagramas seguem entre origens e destinos. A Internet tem muitos protocolos de roteamento. Como vimos na Seção 1.3, a Internet é uma rede de redes e, dentro de uma delas, o administrador pode executar qualquer protocolo de roteamento. Embora a camada de rede contenha o protocolo IP e também numerosos outros de roteamento, ela quase sempre é denominada apenas camada IP, refletindo o fato de que ele é o elemento fundamental que mantém a integridade da Internet.

Camada de enlace

A camada de rede roteia um datagrama por meio de uma série de roteadores entre a origem e o destino. Para levar um pacote de um nó (hospedeiro ou roteador) ao nó seguinte na rota, a camada de rede depende dos serviços da camada de enlace. Em especial, em cada nó, a camada de rede passa o datagrama para a de enlace, que o entrega, ao longo da rota, ao nó seguinte, no qual o datagrama é passado da camada de enlace para a de rede.

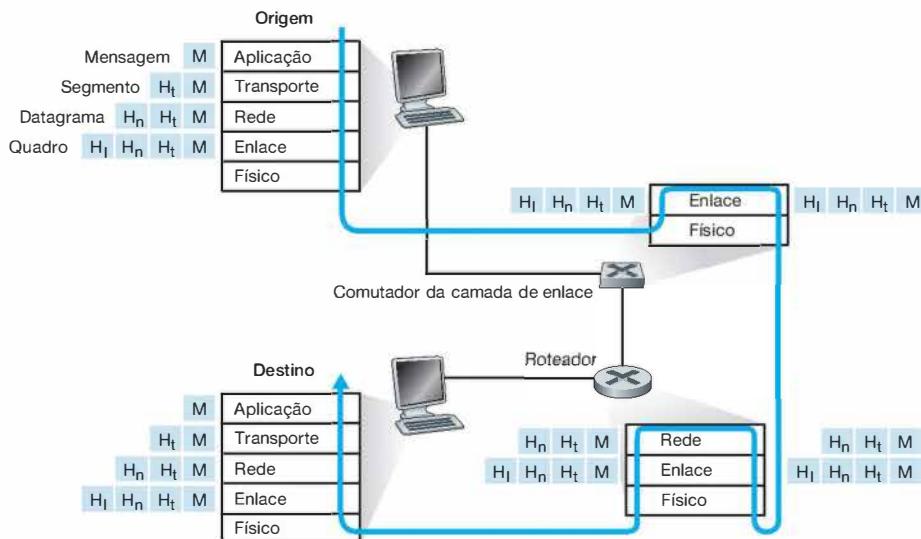
Os serviços prestados pela camada de enlace dependem do protocolo específico empregado no enlace. Por exemplo, alguns desses protocolos proveem entrega garantida entre enlaces, isto é, desde o nó transmissor, passando por um único enlace, até o nó receptor. Note que esse serviço confiável de entrega é diferente do de entrega garantida do TCP, que provê serviço de entrega garantida de um sistema final a outro. Exemplos de protocolos de camadas de enlace são Ethernet, Wi-Fi e o protocolo DOCSIS da rede de acesso por cabo. Como datagramas normalmente precisam transitar por diversos enlaces para irem da origem ao destino, serão manuseados por diferentes protocolos de camada de enlace em diversos enlaces ao longo de sua rota, podendo ser manuseados

por Ethernet em um e por PPP no seguinte. A camada de rede receberá um serviço diferente de cada um dos variados protocolos de camada de enlace. Neste livro, pacotes de camada de enlace serão denominados **quadros**.

Camada física

Enquanto a tarefa da camada de enlace é movimentar quadros inteiros de um elemento da rede até um elemento adjacente, a da camada física é movimentar os *bits individuais* que estão dentro do quadro de um nó para o seguinte. Os protocolos nessa camada de novo dependem do enlace e, além disso, do próprio meio de transmissão do enlace (por exemplo, fios de cobre trançado ou fibra ótica monomodal). Por exemplo, a Ethernet tem muitos protocolos de camada física: um para par de fios de cobre trançado, outro para cabo coaxial, mais um para fibra e assim por diante. Em cada caso, o bit atravessa o enlace de um modo diferente.

O modelo OSI

Após discutir em detalhes a pilha de protocolos da Internet, devemos mencionar que ela não é a única existente. No final dos anos 1970, a Organização Internacional para Padronização (ISO — International Organization for Standardization) propôs que as redes de computadores fossem organizadas em, mais ou menos, sete camadas, denominadas modelo de Interconexão de Sistemas Abertos (OSI — *Open Systems Interconnection*) [ISO, 2012]. O modelo OSI tomou forma quando os protocolos que iriam se tornar protocolos da Internet estavam em sua infância e eram um dos muitos conjuntos em desenvolvimento; na verdade, os inventores do modelo OSI original provavelmente não tinham a Internet em mente ao criá-lo. No entanto, no final dos anos 1970, muitos cursos universitários e de treinamento obtiveram conhecimentos sobre a exigência do ISO e organizaram cursos voltados para o modelo de sete camadas. Em razão de seu impacto precoce na educação de redes, esse modelo continua presente em alguns livros sobre redes e em cursos de treinamento.

As sete camadas do modelo de referência OSI, mostradas na Figura 1.23(b), são: de aplicação, de apresentação, de sessão, de transporte, de rede, de enlace e camada física. A funcionalidade de cinco dessas camadas é a mesma que seus correspondentes da Internet. Desse modo, vamos considerar as duas camadas adicionais presentes no modelo de referência OSI — a de apresentação e a de sessão. O papel da camada de apresentação é prover serviços que permitam que as aplicações de comunicação interpretem o significado dos dados trocados. Entre esses serviços estão a compressão e a codificação de dados (o que não precisa de explicação), assim como a descrição de dados (que, como veremos no Capítulo 9, livram as aplicações da preocupação com o formato interno no qual os dados estão sendo representados/armazenados — formatos que podem ser diferentes de um computador para o outro). A camada de sessão provê a delimitação e sincronização da troca de dados, incluindo os meios de construir um esquema de pontos de verificação e de recuperação.

O fato de a Internet ser desprovida de duas camadas encontradas no modelo de referência OSI faz surgir duas questões: os serviços fornecidos por essas camadas são irrelevantes? E se uma aplicação *precisar* de um desses serviços? A resposta da Internet para essas perguntas é a mesma — depende do desenvolvedor da aplicação. Cabe a ele decidir se um serviço é importante; e se o serviço *for* importante, cabe ao desenvolvedor da aplicação construir essa funcionalidade para ela.

1.5.2 Encapsulamento

A Figura 1.24 apresenta o caminho físico que os dados percorrem: para baixo na pilha de protocolos de um sistema final emissor, para cima e para baixo nas pilhas de protocolos de um comutador e roteador de camada de enlace interveniente, e depois para cima na pilha de protocolos do sistema final receptor. Como discutiremos mais adiante neste livro, ambos, roteadores e comutadores de camada de enlace, são comutadores de pacotes. De modo semelhante a sistemas finais, ambos organizam seu hardware e software de rede em camadas. Mas não implementam *todas* as camadas da pilha de protocolos; em geral executam apenas as camadas de baixo. Como

FIGURA 1.24 HOSPEDEIROS, ROTEADORES E COMUTADORES DE CAMADA DE ENLACE; CADA UM CONTÉM UM CONJUNTO DIFERENTE DE CAMADAS, REFLETINDO SUAS DIFERENÇAS EM FUNCIONALIDADE

ilustra a Figura 1.24, comutadores de camada de enlace realizam as camadas 1 e 2; roteadores executam as camadas 1, 2 e 3. Isso significa, por exemplo, que roteadores da Internet são capazes de executar o protocolo IP (da camada 3), mas comutadores de camada de enlace não. Veremos mais adiante que, embora não reconheçam endereços IP, comutadores de camada de enlace são capazes de reconhecer endereços de camada 2, os da Ethernet. Note que os hospedeiros implementam todas as cinco camadas, o que é consistente com a noção de que a arquitetura da Internet concentra sua complexidade na periferia da rede.

A Figura 1.24 também ilustra o importante conceito de **encapsulamento**. Uma **mensagem da camada de aplicação** na máquina emissora (M na Figura 1.24) é passada para a camada de transporte. No caso mais simples, esta pega a mensagem e anexa informações adicionais (denominadas informações de cabeçalho de camada de transporte, H_t na Figura 1.24) que serão usadas pela camada de transporte do lado receptor. A mensagem da camada de aplicação e as informações de cabeçalho da camada de transporte, juntas, constituem o **segmento da camada de transporte**, que encapsula a mensagem da camada de aplicação. As informações adicionadas podem incluir dados que habilitem a camada de transporte do lado do receptor a entregar a mensagem à aplicação apropriada, além de bits de detecção de erro que permitem que o receptor determine se os bits da mensagem foram modificados em trânsito. A camada de transporte então passa o segmento à camada de rede, que adiciona informações de cabeçalho de camada de rede (H_n na Figura 1.24), como endereços de sistemas finais de origem e de destino, criando um **datagrama de camada de rede**. Este é então passado para a camada de enlace, que (é claro!) adicionará suas próprias informações de cabeçalho e criará um **quadro de camada de enlace**. Assim, vemos que, em cada camada, um pacote possui dois tipos de campos: campos de cabeçalho e um **campo de carga útil**. A carga útil é em geral um pacote da camada acima.

Uma analogia útil que podemos usar aqui é o envio de um memorando entre escritórios de uma empresa pelo correio de uma filial a outra. Suponha que Alice, que está em uma filial, queira enviar um memorando a Bob, que está na outra filial. O *memorando* representa a *mensagem da camada de aplicação*. Alice coloca o memorando em um envelope de correspondência interna em cuja face são escritos o nome e o departamento de Bob. O *envelope de correspondência interna* representa o *segmento da camada de aplicação* — contém as informações de ca-

beçalho (o nome de Bob e seu departamento) e encapsula a mensagem de camada de aplicação (o memorando). Quando a central de correspondência do escritório emissor recebe o envelope, ele é colocado dentro de outro, adequado para envio pelo correio. A central de correspondência emissora também escreve o endereço postal do remetente e do destinatário no envelope postal. Nesse ponto, o *envelope postal* é análogo ao *datagrama* — encapsula o segmento de camada de transporte (o envelope de correspondência interna), que por sua vez encapsula a mensagem original (o memorando). O correio entrega o envelope postal à central de correspondência do escritório destinatário. Nesse local, o processo de desencapsulamento se inicia. A central de correspondência retira o memorando e o encaminha a Bob. Este, por fim, abre o envelope e retira o memorando.

O processo de encapsulamento pode ser mais complexo do que o descrito. Por exemplo, uma mensagem grande pode ser dividida em vários segmentos de camada de transporte (que também podem ser divididos em vários datagramas de camada de rede). Na extremidade receptora, cada segmento deve ser reconstruído a partir dos datagramas que o compõem.

1.6 REDES SOB AMEAÇA

A Internet se tornou essencial para muitas instituições, incluindo empresas grandes e pequenas, universidades e órgãos do governo. Muitas pessoas também contam com a Internet para suas atividades profissionais, sociais e pessoais. Mas atrás de toda essa utilidade e entusiasmo, existe o lado escuro, um lado no qual "vilões" tentam causar problemas em nosso cotidiano danificando nossos computadores conectados à Internet, violando nossa privacidade e tornando inoperantes os serviços da rede dos quais dependemos.

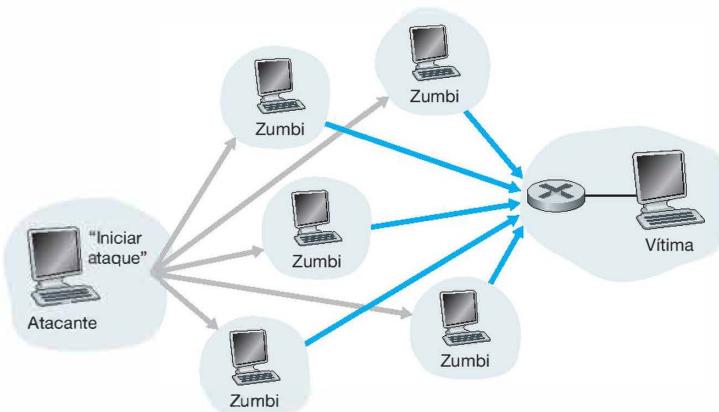
A área de segurança trata de como esses vilões podem ameaçar as redes de computadores e como nós, futuros especialistas no assunto, podemos defender a rede contra essas ameaças ou, melhor ainda, criar novas arquiteturas imunes a tais riscos primeiro. Dadas a frequência e a variedade das ameaças existentes, bem como o perigo de novos e mais destrutivos futuros ataques, a segurança se tornou um assunto principal na área de redes de computadores. Um dos objetivos deste livro é trazer as questões de segurança de rede para o primeiro plano.

Visto que ainda não temos o *know-how* em rede de computadores e em protocolos da Internet, começaremos com uma análise de alguns dos atuais problemas mais predominantes relacionados à segurança. Isto irá aguçar nosso apetite para discussões mais importantes nos capítulos futuros. Começamos com a pergunta: o que pode dar errado? Como as redes de computadores são vulneráveis? Quais são alguns dos tipos de ameaças mais predominantes hoje?

Os vilões podem colocar "*malware*" em seu hospedeiro por meio da Internet

Conectamos aparelhos à Internet porque queremos receber/enviar dados de/para a rede. Isso inclui todos os tipos de recursos vantajosos, como páginas da Web, mensagens de e-mail, MP3, chamadas telefônicas, vídeo em tempo real, resultados de mecanismo de busca etc. Porém, infelizmente, junto com esses recursos vantajosos aparecem os maliciosos — conhecidos de modo coletivo como ***malware*** — que também podem entrar e infectar nossos aparelhos. Uma vez que o *malware* infecta nosso aparelho, ele é capaz de fazer coisas perversas, como apagar nossos arquivos; instalar *spyware* que coleta informações particulares, como nosso número de cartão de crédito, senhas e combinação de teclas, e as envia (pela Internet, é claro!) de volta aos vilões. Nosso hospedeiro comprometido pode estar, também, envolvido em uma rede de milhares de aparelhos comprometidos, conhecidos como ***botnet***, a qual é controlada e utilizada pelos vilões para distribuição de *spams* ou ataques de recusa de serviço distribuídos (que serão discutidos em breve) contra hospedeiros direcionados.

Muitos *malwares* existentes hoje são **autorreprodutivos**: uma vez que infectam um hospedeiro, a partir deste, ele faz a busca por entradas em outros hospedeiros pela Internet, e a dos hospedeiros recém-infectados, procura por entrada em mais hospedeiros. Dessa maneira, o *malware* autorreprodutivo pode se disseminar rapidamente. O *malware* pode se espalhar na forma de um vírus ou um *worm*. Os **vírus** são

malwares que necessitam de uma interação do usuário para infectar seu aparelho. O exemplo clássico é um anexo de e-mail contendo um código executável malicioso. Se o usuário receber e abrir tal anexo, o *malware* será executado em seu aparelho. Geralmente, tais vírus de e-mail se autorreproduzem: uma vez executado, o vírus pode enviar uma mensagem idêntica, com um anexo malicioso idêntico, para, por exemplo, todos os contatos da lista de endereços do usuário. ***Worms*** são *malwares* capazes de entrar em um aparelho sem qualquer interação do usuário. Por exemplo, um usuário pode estar executando uma aplicação de rede frágil para a qual um atacante pode enviar um *malware*. Em alguns casos, sem a intervenção do usuário, a aplicação pode aceitar o *malware* da Internet e executá-lo, criando um *worm*. Este, no aparelho recém-infectado, então, varre a Internet em busca de outros hospedeiros que estejam executando a mesma aplicação de rede vulnerável. Ao encontrá-los, envia uma cópia de si mesmo para eles. Hoje, o *malware* é persuasivo e é caro para se criar uma proteção. À medida que trabalhar com este livro, sugerimos que pense na seguinte questão: o que os projetistas de computadores podem fazer para proteger os aparelhos que utilizam a Internet contra as ameaças de *malware*?

Os vilões podem atacar servidores e infraestrutura de redes

Um amplo grupo de ameaças à segurança pode ser classificado como **ataques de recusa de serviços (DoS — Denial-of-Service)**. Como o nome sugere, um ataque DoS torna uma rede, hospedeiro ou outra parte da infraestrutura inutilizável por usuários verdadeiros. Servidores da Web, de e-mail e DNS (discutidos no Capítulo 2), e redes institucionais podem estar sujeitos aos ataques DoS. Na Internet, esses ataques são extremamente comuns, com milhares deles ocorrendo todo ano [Moore, 2001; Mirkovic, 2005]. A maioria dos ataques DoS na Internet pode ser dividida em três categorias:

- *Ataque de vulnerabilidade*. Envolve o envio de algumas mensagens bem elaboradas a uma aplicação vulnerável ou a um sistema operacional sendo executado em um hospedeiro direcionado. Se a sequência correta de pacotes é enviada a uma aplicação ou sistema operacional vulnerável, o serviço pode parar ou, pior, o hospedeiro pode pifar.
- *Inundação na largura de banda*. O atacante envia um grande número de pacotes ao hospedeiro direcionado — tantos pacotes que o enlace de acesso do alvo se entope, impedindo os pacotes legítimos de alcançarem o servidor.
- *Inundação na conexão*. O atacante estabelece um grande número de conexões TCP semiabertas ou abertas (as conexões TCP são discutidas no Capítulo 3) no hospedeiro-alvo. O hospedeiro pode ficar tão atolado com essas conexões falsas que deixa de aceitar conexões legítimas.

Vamos agora explorar mais detalhadamente o ataque de inundação na largura de banda. Lembrando de nossa análise sobre atraso e perda na Seção 1.4.2, é evidente que se o servidor possui uma taxa de acesso de R bits/s, o atacante precisará enviar tráfego a uma taxa de, mais ou menos, R bits/s para causar dano. Se R for muito grande, uma fonte de ataque única pode não ser capaz de gerar tráfego suficiente para prejudicar o servidor. Além disso, se todo o tráfego emanar de uma fonte única, um roteador mais adiante pode conseguir detectar o ataque e bloquear todo o tráfego da fonte antes que ele se aproxime do servidor. Em um ataque **DoS distribuído (DDoS — Distributed DoS)**, ilustrado na Figura 1.25, o atacante controla múltiplas fontes que sobrecarregam o alvo. Com essa tática, a taxa de tráfego agregada por todas as fontes controladas precisa ser, aproximadamente, R para incapacitar o serviço. Os ataques DDoS que potencializam *botnets* com centenas de hospedeiros comprometidos são uma ocorrência comum hoje em dia [Mirkovic, 2005]. Os ataques DDoS são muito mais difíceis de detectar e de prevenir do que um ataque DoS de um único hospedeiro.

Encorajamos o leitor a considerar a seguinte questão à medida que trabalhar com este livro: o que os projetistas de redes de computadores podem fazer para se protegerem contra ataques DoS? Veremos que são necessárias diferentes defesas para os três tipos de ataques DoS.

FIGURA 1.25 UM ATAQUE DE RECUSA DE SERVIÇO DISTRIBUÍDO (DDoS)

Os vilões podem analisar pacotes

Muitos usuários hoje acessam a Internet por meio de aparelhos sem fio, como laptops conectados à tecnologia Wi-Fi ou aparelhos portáteis com conexões à Internet via telefone celular (abordado no Capítulo 6). Embora o acesso onipresente à Internet seja de extrema conveniência e disponibilize novas aplicações sensacionais aos usuários móveis, ele também cria uma grande vulnerabilidade de segurança — posicionando um receptor passivo nas proximidades do transmissor sem fio, o receptor pode obter uma cópia de cada pacote transmitido! Esses pacotes podem conter todo tipo de informações confidenciais, incluindo senhas, número de identificação, segredos comerciais e mensagens pessoais. Um receptor passivo que grava uma cópia de cada pacote que passa é denominado **analisador de pacote** (*packet sniffer*).

Os analisadores também podem estar distribuídos em ambientes de conexão com fio. Nesses ambientes, como em muitas LANs Ethernet, um analisador de pacote pode obter cópias de todos os pacotes enviados pela LAN. Como descrito na Seção 1.2, as tecnologias de acesso a cabo também transmitem pacotes e são, dessa forma, vulneráveis à análise. Além disso, um vilão que quer ganhar acesso ao roteador de acesso de uma instituição ou enlace de acesso para a Internet pode instalar um analisador que faça uma cópia de cada pacote que vai para/de a empresa. Os pacotes farejados podem, então, ser analisados *off-line* em busca de informações confidenciais.

O software para analisar pacotes está disponível gratuitamente em diversos sites da Internet e em produtos comerciais. Professores que ministram um curso de redes passam exercícios que envolvem a escrita de um programa de reconstrução de dados da camada de aplicação e um programa analisador de pacotes. De fato, os Wireshark labs [Wireshark, 2012] associados a este texto (veja o Wireshark lab introdutório ao final deste capítulo) utilizam exatamente tal analisador de pacotes.

Como os analisadores de pacote são passivos — ou seja, não introduzem pacotes no canal —, eles são difíceis de detectar. Portanto, quando enviamos pacotes para um canal sem fio, devemos aceitar a possibilidade de que alguém possa estar copiando nossos pacotes. Como você deve ter imaginado, uma das melhores defesas contra a análise de pacote envolve a criptografia, que será explicada no Capítulo 8, já que se aplica à segurança de rede.

Os vilões podem se passar por alguém de sua confiança

Por incrível que pareça, é facílimo (*você* saberá como fazer isso à medida que ler este livro!) criar um pacote com qualquer endereço de origem, conteúdo de pacote e endereço de destino e, depois, transmiti-lo para a Internet, que, obedientemente, o encaminhará ao destino. Imagine que um receptor inocente (digamos, um roteador da Internet) que recebe tal pacote acredita que o endereço de origem (falso) seja confiável e então executa um

comando integrado ao conteúdo do pacote (digamos, que modifica sua base de encaminhamento). A capacidade de introduzir pacotes na Internet com um endereço de origem falso é conhecida como **IP *spoofing***, e é uma das muitas maneiras pelas quais o usuário pode se passar por outro.

Para resolver esse problema, precisaremos de uma *autenticação do ponto final*, ou seja, um mecanismo que nos permita determinar com certeza se uma mensagem se origina de onde pensamos. Mais uma vez, sugerimos que pense em como isso pode ser feito em aplicações de rede e protocolos à medida que avança sua leitura pelos capítulos deste livro. Exploraremos mais mecanismos para comprovação da fonte no Capítulo 8.

Ao encerrar esta seção, deve-se considerar como a Internet se tornou um local inseguro, antes de tudo. A resposta breve é que a Internet foi, a princípio, criada dessa maneira, baseada no modelo de "um grupo de usuários de confiança mútua ligados a uma rede transparente" [Blumenthal, 2001] — um modelo no qual (por definição) não há necessidade de segurança. Muitos aspectos da arquitetura inicial da Internet refletem profundamente essa noção de confiança mútua. Por exemplo, a capacidade de um usuário enviar um pacote a qualquer outro é mais uma falha do que um recurso solicitado/concedido, e acredita-se piamente na identidade do usuário, em vez de ela ser autenticada como padrão.

Mas a Internet de hoje decerto não envolve "usuários de confiança mútua". Contudo, os usuários atuais ainda precisam se comunicar mesmo quando não confiam um no outro, podem querer se comunicar de modo anônimo, podem se comunicar indiretamente por terceiros (por exemplo, Web *caches*, que serão estudados no Capítulo 2, ou agentes móveis para assistência, que serão estudados no Capítulo 6), e podem desconfiar do hardware, software e até mesmo do ar pelo qual eles se comunicam. Temos agora muitos desafios relacionados à segurança perante nós à medida que prosseguimos com o livro: devemos buscar proteção contra a análise, disfarce da origem, ataques *man-in-the-middle*, ataques DDoS, *malware* e outros. Precisamos manter em mente que a comunicação entre usuários de confiança mútua é mais uma exceção do que uma regra. Seja bem-vindo ao mundo da moderna rede de computadores!

1.7 HISTÓRIA DAS REDES DE COMPUTADORES E DA INTERNET

Da Seção 1.1 à 1.6, apresentamos um panorama da tecnologia de redes de computadores e da Internet. Agora, você já deve saber o suficiente para impressionar sua família e amigos! Contudo, se quiser ser mesmo o maior sucesso na próxima festa, você deve rechear seu discurso com pérolas da fascinante história da Internet [Segaller, 1998].

1.7.1 Desenvolvimento da comutação de pacotes: 1961-1972

Os primeiros passos da disciplina de redes de computadores e da Internet atual podem ser traçados desde o início da década de 1960, quando a rede telefônica era a rede de comunicação dominante no mundo inteiro. Lembre-se de que na Seção 1.3 dissemos que a rede de telefonia usa comutação de circuitos para transmitir informações de uma origem a um destino — uma escolha acertada, já que a voz é transmitida a uma taxa constante entre os pontos. Dada a importância cada vez maior dos computadores no início da década de 1960 e o advento de computadores com tempo compartilhado, nada seria mais natural do que considerar a questão de como interligar computadores para que pudessem ser compartilhados entre usuários geograficamente dispersos. O tráfego gerado por esses usuários provavelmente era feito por *rajadas* — períodos de atividade, como o envio de um comando a um computador remoto, seguidos de períodos de inatividade, como a espera por uma resposta ou o exame de uma resposta recebida.

Três grupos de pesquisa ao redor do mundo, sem que nenhum tivesse conhecimento do trabalho do outro [Leiner, 1998], começaram a inventar a comutação de pacotes como uma alternativa poderosa e eficiente à comutação de circuitos. O primeiro trabalho publicado sobre técnicas de comutação de pacotes foi o de Leonard Kleinrock [Kleinrock, 1961, 1964], que, naquela época, era um aluno de graduação no MIT. Usando a teoria de

filas, seu trabalho demonstrou, com elegância, a eficácia da abordagem da comutação de pacotes para fontes de tráfego intermitentes (em rajadas). Em 1964, Paul Baran [Baran, 1964], do Rand Institute, começou a investigar a utilização de comutação de pacotes na transmissão segura de voz pelas redes militares, ao mesmo tempo que Donald Davies e Roger Scantlebury desenvolviam suas ideias sobre esse assunto no National Physical Laboratory, na Inglaterra.

Os trabalhos desenvolvidos no MIT, no Rand Institute e no NPL foram os alicerces do que hoje é a Internet. Mas a Internet tem uma longa história de atitudes do tipo "construir e demonstrar", que também data do início da década de 1960. J. C. R. Licklider [DEC, 1990] e Lawrence Roberts, ambos colegas de Kleinrock no MIT, foram adiante e lideraram o programa de ciência de computadores na ARPA (Advanced Research Projects Agency — Agência de Projetos de Pesquisa Avançada), nos Estados Unidos. Roberts publicou um plano geral para a ARPAnet [Roberts, 1967], a primeira rede de computadores por comutação de pacotes e uma ancestral direta da Internet pública de hoje. Em 1969, no Dia do Trabalho nos Estados Unidos, foi instalado o primeiro comutador de pacotes na UCLA (Universidade da Califórnia em Los Angeles) sob a supervisão de Kleinrock. Pouco tempo depois, foram instalados três comutadores de pacotes adicionais no Stanford Research Institute (SRI), na Universidade da Califórnia em Santa Bárbara e na Universidade de Utah (Figura 1.26). O incipiente precursor da Internet tinha quatro nós no final de 1969. Kleinrock recorda que a primeiríssima utilização da rede foi fazer um login remoto entre a UCLA e o SRI, derrubando o sistema [Kleinrock, 2004].

Em 1972, a ARPAnet tinha cerca de 15 nós e foi apresentada publicamente pela primeira vez por Robert Kahn. O primeiro protocolo fim a fim entre sistemas finais da ARPAnet, conhecido como protocolo de controle de rede (*network-control protocol* — NCP), estava concluído [RFC 001]. Com um protocolo fim a fim à disposição, a escrita de aplicações tornou-se possível. Em 1972, Ray Tomlinson, da BBN, escreveu o primeiro programa de e-mail.

FIGURA 1.26 UM DOS PRIMEIROS COMUTADORES DE PACOTES

1.7.2 Redes proprietárias e trabalho em rede: 1972-1980

A ARPAnet inicial era uma rede isolada, fechada. Para se comunicar com uma máquina da ARPAnet, era preciso estar ligado a um outro IMP dessa rede. Do início a meados de 1970, surgiram novas redes independentes de comutação de pacotes: ALOHAnet, uma rede de micro-ondas ligando universidades das ilhas do Havaí [Abramson, 1970], bem como as redes de pacotes por satélite [RFC 829] e por rádio [Kahn, 1978] da DARPA; Telenet, uma rede comercial de comutação de pacotes da BBN baseada na tecnologia ARPAnet; Cyclades, uma rede de comutação de pacotes pioneira na França, montada por Louis Pouzin [Think, 2002]; redes de tempo compartilhado como a Tymnet e a rede GE Information Services, entre outras que surgiram no final da década de 1960 e início da década de 1970 [Schwartz, 1977]; rede SNA da IBM (1969–1974), cujo trabalho comparava-se ao da ARPAnet [Schwartz, 1977].

O número de redes estava crescendo. Hoje, com perfeita visão do passado, podemos perceber que aquela era a hora certa para desenvolver uma arquitetura abrangente para conectar redes. O trabalho pioneiro de interconexão de redes, sob o patrocínio da DARPA (Defense Advanced Research Projects Agency — Agência de Projetos de Pesquisa Avançada de Defesa), criou basicamente *uma rede de redes* e foi realizado por Vinton Cerf e Robert Kahn [Cerf, 1974]; o termo *internetting* foi cunhado para descrever esse trabalho.

Esses princípios de arquitetura foram incorporados ao TCP. As primeiras versões desse protocolo, contudo, eram muito diferentes do TCP de hoje. Elas combinavam uma entrega sequencial confiável de dados via retransmissão por sistema final (que ainda faz parte do TCP de hoje) com funções de envio (que hoje são desempenhadas pelo IP). As primeiras experiências com o TCP, combinadas com o reconhecimento da importância de um serviço de transporte fim a fim não confiável, sem controle de fluxo, para aplicações como voz em pacotes, levaram à separação entre IP e TCP e ao desenvolvimento do protocolo UDP. Os três protocolos fundamentais da Internet que temos hoje — TCP, UDP e IP — estavam conceitualmente disponíveis no final da década de 1970.

Além das pesquisas sobre a Internet realizadas pela DARPA, muitas outras atividades importantes relacionadas ao trabalho em rede estavam em andamento. No Havaí, Norman Abramson estava desenvolvendo a ALOHAnet, uma rede de pacotes por rádio que permitia que vários lugares remotos das ilhas havaianas se comunicassem entre si. O ALOHA [Abramson, 1970] foi o primeiro protocolo de acesso múltiplo que permitiu que usuários geograficamente dispersos compartilhassem um único meio de comunicação *broadcast* (uma frequência de rádio). Metcalfe e Boggs se basearam no trabalho de Abramson sobre protocolo de múltiplo acesso quando desenvolveram o protocolo Ethernet [Metcalfe, 1976] para redes compartilhadas de transmissão broadcast por fio. O interessante é que o protocolo Ethernet de Metcalfe e Boggs foi motivado pela necessidade de conectar vários PCs, impressoras e discos compartilhados [Perkins, 1994]. Há 25 anos, bem antes da revolução do PC e da explosão das redes, Metcalfe e Boggs estavam lançando as bases para as LANs de PCs de hoje.

1.7.3 Proliferação de redes: 1980-1990

Ao final da década de 1970, cerca de 200 máquinas estavam conectadas à ARPAnet. Ao final da década de 1980, o número de máquinas ligadas à Internet pública, uma confederação de redes muito parecida com a Internet de hoje, alcançaria cem mil. A década de 1980 seria uma época de formidável crescimento.

Grande parte daquele crescimento foi consequência de vários esforços distintos para criar redes de computadores para interligar universidades. A BITNET processava e-mails e fazia transferência de arquivos entre diversas universidades do nordeste dos Estados Unidos. A CSNET (Computer Science NETwork — rede da ciência de computadores) foi formada para interligar pesquisadores de universidades que não tinham acesso à ARPAnet. Em 1986, foi criada a NSFNET para prover acesso a centros de supercomputação patrocinados pela NSF. Partindo de uma velocidade inicial de 56 kbits/s, ao final da década o *backbone* da NSFNET estaria funcionando a 1,5 Mbits/s e servindo como *backbone* primário para a interligação de redes regionais.

Na comunidade da ARPAnet, já estavam sendo encaixados muitos dos componentes finais da arquitetura da Internet de hoje. No dia 1º de janeiro de 1983, o TCP/IP foi adotado oficialmente como o novo padrão de

protocolo de máquinas para a ARPAnet (em substituição ao protocolo NCP). Pela importância do evento, o dia da transição do NCP para o TCP/IP [RFC 801] foi marcado com antecedência — a partir daquele dia todas as máquinas tiveram de adotar o TCP/IP. No final da década de 1980, foram agregadas importantes extensões ao TCP para implementação do controle de congestionamento baseado em hospedeiros [Jacobson, 1988]. Também foi desenvolvido o sistema de nomes de domínios (DNS) utilizado para mapear nomes da Internet fáceis de entender (por exemplo, gaia.cs.umass.edu) para seus endereços IP de 32 bits [RFC 1034].

Em paralelo ao desenvolvimento da ARPAnet (que em sua maior parte deve-se aos Estados Unidos), no início da década de 1980 os franceses lançaram o projeto Minitel, um plano ambicioso para levar as redes de dados para todos os lares. Patrocinado pelo governo francês, o sistema consistia em uma rede pública de comutação de pacotes (baseada no conjunto de protocolos X.25, que usava circuitos virtuais), servidores Minitel e terminais baratos com modems de baixa velocidade embutidos. O Minitel transformou-se em um enorme sucesso em 1984, quando o governo francês forneceu, gratuitamente, um terminal para toda residência francesa que quisesse. O sistema incluía sites de livre acesso — como o da lista telefônica — e também particulares, que cobravam uma taxa de cada usuário baseada no tempo de utilização. No seu auge, em meados de 1990, o Minitel oferecia mais de 20 mil serviços, que iam desde *home banking* até bancos de dados especializados para pesquisa. Estava presente em grande parte dos lares franceses dez anos antes sequer de a maioria dos norte-americanos ouvir falar de Internet.

1.7.4 A explosão da Internet: a década de 1990

A década de 1990 estreou com vários eventos que simbolizaram a evolução contínua e a comercialização iminente da Internet. A ARPAnet, a progenitora da Internet, deixou de existir. Em 1991, a NSFNET extinguiu as restrições que impunha à sua utilização com finalidades comerciais, mas, em 1995, perderia seu mandato quando o tráfego de *backbone* da Internet passou a ser carregado por provedores de serviços.

O principal evento da década de 1990, no entanto, foi o surgimento da World Wide Web, que levou a Internet para os lares e as empresas de milhões de pessoas no mundo inteiro. A Web serviu também como plataforma para a habilitação e a disponibilização de centenas de novas aplicações, inclusive busca (por exemplo, Google e Bing), comércio pela Internet (por exemplo, Amazon e eBay) e redes sociais (por exemplo, Facebook).

A Web foi inventada no CERN (European Center for Nuclear Physics — Centro Europeu para Física Nuclear) por Tim Berners-Lee entre 1989 e 1991 [Berners-Lee, 1989], com base em ideias originadas de trabalhos anteriores sobre hipertexto realizados por Vannevar Bush [Bush, 1945], na década de 1940, e por Ted Nelson [Xanadu, 2012], na década de 1960. Berners-Lee e seus companheiros desenvolveram versões iniciais de HTML, HTTP, um servidor Web e um navegador (*browser*) — os quatro componentes fundamentais da Web. Por volta de 1993, havia cerca de 200 servidores Web em operação, e esse conjunto era apenas um prenúncio do que estava por vir. Nessa época, vários pesquisadores estavam desenvolvendo navegadores Web com interfaces GUI (Graphical User Interface — interface gráfica de usuário), entre eles Marc Andreessen, que liderou o desenvolvimento do popular navegador Mosaic, junto com Kim Clark, que formaram a Mosaic Communications, que mais tarde se transformou na Netscape Communications Corporation [Cusumano, 1998; Quittner, 1998]. Em 1995, estudantes universitários estavam usando navegadores Mosaic e Netscape para navegar na Web diariamente. Na época, empresas — grandes e pequenas — começaram a operar servidores e a realizar transações comerciais pela Web. Em 1996, a Microsoft começou a desenvolver navegadores, dando início à guerra entre Netscape e Microsoft, vencida pela última alguns anos mais tarde [Cusumano, 1998].

A segunda metade da década de 1990 foi um período de tremendo crescimento e inovação, com grandes corporações e milhares de novas empresas criando produtos e serviços para a Internet. No final do milênio a Internet dava suporte a centenas de aplicações populares, entre elas quatro de enorme sucesso:

- e-mail, incluindo anexos e correio eletrônico com acesso pela Web;
- a Web, incluindo navegação pela Web e comércio pela Internet;
- serviço de mensagem instantânea, com listas de contato;
- compartilhamento *peer-to-peer* de arquivos MP3, cujo pioneiro foi o Napster.

O interessante é que as duas primeiras dessas aplicações de sucesso arrasador vieram da comunidade de pesquisas, ao passo que as duas últimas foram criadas por alguns jovens empreendedores.

No período de 1995 a 2001, a Internet realizou uma viagem vertiginosa nos mercados financeiros. Antes mesmo de se mostrarem lucrativas, centenas de novas empresas faziam suas ofertas públicas iniciais de ações e começavam a ser negociadas em bolsas de valores. Muitas empresas eram avaliadas em bilhões de dólares sem ter nenhum fluxo significativo de receita. As ações da Internet sofreram uma queda também vertiginosa em 2000-2001, e muitas novas empresas fecharam. Não obstante, várias outras surgiram como grandes vencedoras no mundo da Internet, entre elas Microsoft, Cisco, Yahoo, eBay, Google e Amazon.

1.7.5 O novo milênio

A inovação na área de redes de computadores continua a passos largos. Há progressos em todas as frentes, incluindo distribuição de roteadores mais velozes e velocidades de transmissão mais altas nas redes de acesso e nos *backbones* da rede. Mas os seguintes desenvolvimentos merecem atenção especial:

- Desde o início do milênio, vimos a implementação agressiva do acesso à Internet por banda larga nos lares — não apenas modems a cabo e DSL, mas também "*fiber to the home*", conforme discutimos na Seção 1.2. Esse acesso à Internet de alta velocidade preparou a cena para uma série de aplicações de vídeo, incluindo a distribuição de vídeo gerado pelo usuário (por exemplo, YouTube), fluxo contínuo por demanda de filmes e shows de televisão (por exemplo, Netflix) e videoconferência entre várias pessoas (por exemplo, Skype).
- A onipresença cada vez maior das redes Wi-Fi públicas de alta velocidade (54 Mbits/s e mais altas) e o acesso à Internet com velocidade média (até alguns Mbits/s) por redes de telefonia celular 3G e 4G não apenas está possibilitando permanecer constantemente conectado enquanto se desloca, mas também permite novas aplicações específicas à localização. O número de dispositivos sem fio conectados ultrapassou o número de dispositivos com fio em 2011. Esse acesso sem fio em alta velocidade preparou a cena para o rápido surgimento de computadores portáteis (iPhones, Androids, iPads etc.), que possuem acesso constante e livre à Internet.
- Redes sociais on-line, como Facebook e Twitter, criaram redes de pessoas maciças em cima da Internet. Muitos usuários hoje "vivem" principalmente dentro do Facebook. Através de suas APIs, as redes sociais on-line criam plataformas para novas aplicações em rede e jogos distribuídos.
- Conforme discutimos na Seção 1.3.3, os provedores de serviços on-line, como Google e Microsoft, implementaram suas próprias amplas redes privativas, que não apenas conectam seus centros de dados distribuídos em todo o planeta, mas são usadas para evitar a Internet ao máximo possível, emparelhando diretamente com ISPs de nível mais baixo. Como resultado, Google oferece resultados de busca e acesso a e-mail quase instantaneamente, como se seus centros de dados estivessem rodando dentro do computador de cada usuário.
- Muitas empresas de comércio na Internet agora estão rodando suas aplicações na "nuvem" — como na EC2 da Amazon, na Application Engine da Google ou na Azure da Microsoft. Diversas empresas e universidades também migraram suas aplicações da Internet (por exemplo, e-mail e hospedagem de páginas Web) para a nuvem. Empresas de nuvem não apenas oferecem ambientes de computação e armazenamento escaláveis às aplicações, mas também lhes oferecem acesso implícito às suas redes privativas de alto desempenho.

1.8 RESUMO

Neste capítulo, abordamos uma quantidade imensa de assuntos. Examinamos as várias peças de hardware e software que compõem a Internet, em especial, e redes de computadores, em geral. Começamos pela periferia

da rede, examinando sistemas finais e aplicações, além do serviço de transporte fornecido às aplicações que executam nos sistemas finais. Também vimos as tecnologias de camada de enlace e meio físico encontrados na rede de acesso. Em seguida, mergulhamos no interior da rede e chegamos ao seu núcleo, identificando comutação de pacotes e comutação de circuitos como as duas abordagens básicas do transporte de dados por uma rede de telecomunicações, expondo os pontos fortes e fracos de cada uma delas. Examinamos, então, as partes inferiores (do ponto de vista da arquitetura) da rede — as tecnologias de camada de enlace e os meios físicos comumente encontrados na rede de acesso. Estudamos também a estrutura da Internet global e aprendemos que ela é uma rede de redes. Vimos que a estrutura hierárquica da Internet, composta de ISPs de níveis mais altos e mais baixos, permitiu que ela se expandisse e incluísse milhares de redes.

Na segunda parte deste capítulo introdutório, abordamos diversos tópicos fundamentais da área de redes de computadores. Primeiro examinamos as causas de atrasos e perdas de pacotes em uma rede de comutação de pacotes. Desenvolvemos modelos quantitativos simples de atrasos de transmissão, de propagação e de fila, bem como modelos de vazão; esses modelos de atrasos serão muito usados nos problemas propostos em todo o livro. Em seguida examinamos camadas de protocolo e modelos de serviço, princípios fundamentais de arquitetura de redes aos quais voltaremos a nos referir neste livro. Analisamos também alguns dos ataques mais comuns na Internet. Terminamos nossa introdução sobre redes com uma breve história das redes de computadores. O primeiro capítulo constitui um minicurso sobre redes de computadores.

Portanto, percorremos de fato um extraordinário caminho neste primeiro capítulo! Se você estiver um pouco assustado, não se preocupe. Abordaremos todas essas ideias em detalhes nos capítulos seguintes (é uma promessa, e não uma ameaça!). Por enquanto, esperamos que, ao encerrar este capítulo, você tenha adquirido uma noção, ainda que incipiente, das partes que formam uma rede, um domínio ainda em desenvolvimento do vocabulário de redes (não se acanhe de voltar aqui para consulta) e um desejo cada vez maior de aprender mais sobre elas. Essa é a tarefa que nos espera no restante deste livro.

O guia deste livro

Antes de iniciarmos qualquer viagem, sempre é bom consultar um guia para nos familiarizar com as estradas principais e desvios que encontraremos pela frente. O destino final da viagem que estamos prestes a empreender é um entendimento profundo do como, do quê e do porquê das redes de computadores. Nosso guia é a sequência de capítulos:

1. Redes de computadores e a Internet
2. Camada de aplicação
3. Camada de transporte
4. Camada de rede
5. Camada de enlace e redes locais (LANs)
6. Redes sem fio e móveis
7. Redes multimídia
8. Segurança em redes de computadores
9. Gerenciamento de rede

Os Capítulos 2 a 5 são os quatro capítulos centrais deste livro. Note que eles estão organizados segundo as quatro camadas superiores da pilha de cinco camadas de protocolos da Internet, com um capítulo para cada uma. Note também que nossa jornada começará no topo da pilha, a saber, a camada de aplicação, e prosseguirá daí para baixo. O princípio racional que orienta essa jornada de cima para baixo é que, entendidas as aplicações, podemos compreender os serviços de rede necessários para dar-lhes suporte. Então, poderemos examinar, um por um, os vários modos como esses serviços poderiam ser executados por uma arquitetura de rede. Assim, o estudo das aplicações logo no início dá motivação para o restante do livro.

A segunda metade — Capítulos 6 a 9 — aborda quatro tópicos de extrema importância (e de certa maneira independentes) de redes modernas. No Capítulo 6, examinamos as redes sem fio e móvel, incluindo LANs sem fio (incluindo Wi-Fi e Bluetooth), redes de telefonia celular (GSM) e mobilidade (nas redes IP e GSM). No Capítulo 7 (Redes multimídia), examinamos aplicações de áudio e vídeo, como telefone por Internet, videoconferência e fluxo contínuo de mídia armazenada. Examinamos também como uma rede de comutação de pacotes pode ser projetada para prover qualidade de serviço consistente para aplicações de áudio e vídeo. No Capítulo 8 (Segurança em redes de computadores), analisamos, primeiro, os fundamentos da criptografia e da segurança de redes e, em seguida, de que modo a teoria básica está sendo aplicada a um amplo leque de contextos da Internet. No último capítulo ("Gerenciamento de redes"), examinamos as questões fundamentais do gerenciamento de redes, bem como os principais protocolos da Internet utilizados para esse fim.

EXERCÍCIOS DE FIXAÇÃO E PERGUNTAS

Questões de revisão do Capítulo 1

SEÇÃO 1.1

R1. Qual é a diferença entre um hospedeiro e um sistema final? Cite os tipos de sistemas finais. Um servidor Web é um sistema final?

R2. A palavra *protocolo* é muito usada para descrever relações diplomáticas. Como a Wikipedia descreve um protocolo diplomático?

R3. Por que os padrões são importantes para os protocolos?

SEÇÃO 1.2

R4. Cite seis tecnologias de acesso. Classifique cada uma delas nas categorias acesso residencial, acesso corporativo ou acesso móvel.

R5. A taxa de transmissão HFC é dedicada ou é compartilhada entre usuários? É possível haver colisões na direção provedor-usuário de um canal HFC? Por quê?

R6. Cite as tecnologias de acesso residencial disponíveis em sua cidade. Para cada tipo de acesso, apresente a taxa *downstream*, a taxa *upstream* e o preço mensal anunciados.

R7. Qual é a taxa de transmissão de LANs Ethernet?

R8. Cite alguns meios físicos utilizados para instalar a Ethernet.

R9. Modems discados, HFC, DSL e FTTH são usados para acesso residencial. Para cada uma dessas tecnologias de acesso, cite uma faixa de taxas de transmissão e comente se a taxa de transmissão é compartilhada ou dedicada.

R10. Descreva as tecnologias de acesso sem fio mais populares atualmente. Faça uma comparação entre elas.

SEÇÃO 1.3

R11. Suponha que exista exatamente um comutador de pacotes entre um computador de origem e um de destino. As taxas de transmissão entre a máquina de origem e o comutador e entre este e a máquina de destino são R_1 e R_2, respectivamente. Admitindo que um roteador use comutação de pacotes do tipo armazena-e-reenvia, qual é o atraso total fim a fim para enviar um pacote de comprimento L? (Desconsidere formação de fila, atraso de propagação e atraso de processamento.)

R12. Qual é a vantagem de uma rede de comutação de circuitos em relação a uma de comutação de pacotes? Quais são as vantagens da TDM sobre a FDM em uma rede de comutação de circuitos?

R13. Suponha que usuários compartilhem um enlace de 2 Mbits/s e que cada usuário transmita continuamente a 1 Mbit/s, mas cada um deles transmite apenas 20% do tempo. (Veja a discussão sobre multiplexação estatística na Seção 1.3.)

a. Quando a comutação de circuitos é utilizada, quantos usuários podem ser admitidos?

b. Para o restante deste problema, suponha que seja utilizada a comutação de pacotes. Por que não haverá atraso de fila antes de um enlace se dois ou menos usuários transmitirem ao mesmo tempo? Por que haverá atraso de fila se três usuários transmitirem ao mesmo tempo?

c. Determine a probabilidade de um dado usuário estar transmitindo.

d. Suponha agora que haja três usuários. Determine a probabilidade de, a qualquer momento, os três usuários transmitirem simultaneamente. Determine a fração de tempo durante o qual a fila cresce.

R14. Por que dois ISPs no mesmo nível de hierarquia farão emparelhamento? Como um IXP consegue ter lucro?

R15. Alguns provedores de conteúdo criaram suas próprias redes. Descreva a rede da Google. O que motiva os provedores de conteúdo a criar essas redes?

SEÇÃO 1.4

R16. Considere o envio de um pacote de uma máquina de origem a uma de destino por uma rota fixa. Relacione os componentes do atraso que formam o atraso fim a fim. Quais deles são constantes e quais são variáveis?

R17. Visite o applet "Transmission *versus* Propagation Delay" no site de apoio do livro. Entre as taxas, o atraso de propagação e os tamanhos de pacote disponíveis, determine uma combinação para a qual o emissor termine de transmitir antes que o primeiro bit do pacote chegue ao receptor. Ache outra combinação para a qual o primeiro bit do pacote alcança o receptor antes que o emissor termine de transmitir.

R18. Quanto tempo um pacote de 1.000 bytes leva para se propagar através de um enlace de 2.500 km de distância, com uma velocidade de propagação de 2,5 · 108 m/s e uma taxa de transmissão de 2 Mbits/s? Em geral, quanto tempo um pacote de comprimento *L* leva para se propagar através de um enlace de distância *d*, velocidade de propagação *s*, e taxa de transmissão de *R* bits/s? Esse atraso depende do comprimento do pacote? Depende da taxa de transmissão?

R19. Suponha que o hospedeiro A queira enviar um arquivo grande para o hospedeiro B. O percurso de A para B possui três enlaces, de taxas R_1 = 500 kbits/s, R_2 = 2 Mbits/s, e R_3 = 1 Mbit/s.

a. Considerando que não haja nenhum outro tráfego na rede, qual é a vazão para a transferência de arquivo?

b. Suponha que o arquivo tenha 4 milhões de bytes. Dividindo o tamanho do arquivo pela vazão, quanto tempo levará a transferência para o hospedeiro B?

c. Repita os itens "a" e "b", mas agora com R_2 reduzido a 100 kbits/s.

R20. Suponha que o sistema final A queira enviar um arquivo grande para o sistema B. Em um nível muito alto, descreva como o sistema A cria pacotes a partir do arquivo. Quando um desses arquivos chega ao comutador de pacote, quais informações no pacote o comutador utiliza para determinar o enlace através do qual o pacote é encaminhado? Por que a comutação de pacotes na Internet é semelhante a dirigir de uma cidade para outra pedindo informações ao longo do caminho?

R21. Visite o applet "Queuing and Loss" no site de apoio do livro. Qual é a taxa de emissão máxima e a taxa de transmissão mínima? Com essas taxas, qual é a intensidade do tráfego? Execute o applet com essas taxas e determine o tempo que leva a ocorrência de uma perda de pacote. Repita o procedimento mais uma vez e determine de novo o tempo de ocorrência para a perda de pacote. Os resultados são diferentes? Por quê? Por que não?

SEÇÃO 1.5

R22. Cite cinco tarefas que uma camada pode executar. É possível que uma (ou mais) dessas tarefas seja(m) realizada(s) por duas (ou mais) camadas?

R23. Quais são as cinco camadas da pilha de protocolo da Internet? Quais as principais responsabilidades de cada uma dessas camadas?

R24. O que é uma mensagem de camada de aplicação? Um segmento de camada de transporte? Um datagrama de camada de rede? Um quadro de camada de enlace?

R25. Que camadas da pilha do protocolo da Internet um roteador processa? Que camadas um comutador de camada de enlace processa? Que camadas um sistema final processa?

SEÇÃO 1.6

R26. Qual é a diferença entre um vírus e um *worm*?

R27. Descreva como pode ser criado uma *botnet* e como ela pode ser utilizada no ataque DDoS.

R28. Suponha que Alice e Bob estejam enviando pacotes um para o outro por uma rede de computadores e que Trudy se posicione na rede para poder capturar todos os pacotes enviados por Alice e enviar o que quiser para Bob; ela também consegue capturar todos os pacotes enviados por Bob e enviar o que quiser para Alice. Cite algumas atitudes maliciosas que Trudy pode fazer a partir de sua posição.

PROBLEMAS

P1. Projete e descreva um protocolo de nível de aplicação para ser usado entre um caixa eletrônico e o computador central de um banco. Esse protocolo deve permitir verificação do cartão e da senha de um usuário, consulta do saldo de sua conta (que é mantido no computador central) e saque de dinheiro (isto é, entrega de dinheiro ao usuário). As entidades do protocolo devem estar preparadas para resolver o caso comum em que não há dinheiro suficiente na conta para cobrir o saque. Especifique seu protocolo relacionando as mensagens trocadas e as ações realizadas pelo caixa automático ou pelo computador central do banco na transmissão e recepção de mensagens. Esquematize a operação de seu protocolo para o caso de um saque simples sem erros, usando um diagrama semelhante ao da Figura 1.2. Descreva explicitamente o que seu protocolo espera do serviço de transporte fim a fim.

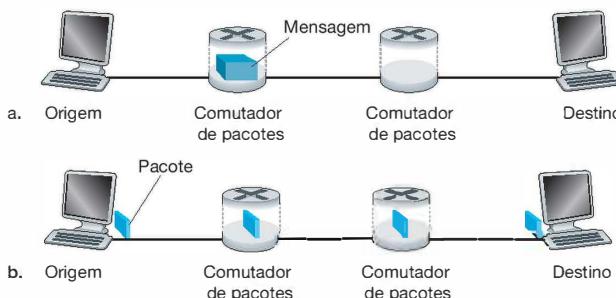
P2. A Equação 1.1 contém uma fórmula para o atraso fim a fim do envio de um pacote de comprimento L por N enlaces com taxa de transmissão R. Generalize essa fórmula para enviar P desses pacotes de ponta a ponta pelos N enlaces.

P3. Considere uma aplicação que transmita dados a uma taxa constante (por exemplo, a origem gera uma unidade de dados de N bits a cada k unidades de tempo, onde k é pequeno e fixo). Considere também que, quando essa aplicação começa, continuará em funcionamento por um período de tempo relativamente longo. Responda às seguintes perguntas, dando uma breve justificativa para suas respostas:

a. O que seria mais apropriado para essa aplicação: uma rede de comutação de circuitos ou uma rede de comutação de pacotes? Por quê?

b. Suponha que seja usada uma rede de comutação de pacotes e que o único tráfego venha de aplicações como a descrita anteriormente. Além disso, imagine que a soma das velocidades de dados da aplicação seja menor do que a capacidade de cada enlace. Será necessário algum tipo de controle de congestionamento? Por quê?

P4. Considere a rede de comutação de circuitos da Figura 1.13. Lembre-se de que há 4 circuitos em cada enlace. Rotule os quatro comutadores A, B, C e D, seguindo no sentido horário.

a. Qual é o número máximo de conexões simultâneas que podem estar em curso a qualquer instante nessa rede?

b. Suponha que todas as conexões sejam entre os comutadores A e C. Qual é o número máximo de conexões simultâneas que podem estar em curso?

c. Suponha que queiramos fazer quatro conexões entre os comutadores A e C, e outras quatro conexões entre os *switches* B e D. Podemos rotear essas chamadas pelos quatro enlaces para acomodar todas as oito conexões?

P5. Considere novamente a analogia do comboio de carros da Seção 1.4. Admita uma velocidade de propagação de 100 km/h.

a. Suponha que o comboio viaje 150 km, começando em frente ao primeiro dos postos de pedágio, passando por um segundo e terminando após um terceiro. Qual é o atraso fim a fim?

b. Repita o item 'a' admitindo agora que haja oito carros no comboio em vez de dez.

P6. Este problema elementar começa a explorar atrasos de propagação e de transmissão, dois conceitos centrais em redes de computadores. Considere dois hospedeiros, A e B, conectados por um único enlace de taxa R bits/s. Suponha que eles estejam separados por m metros e que a velocidade de propagação ao longo do enlace seja de s metros/segundo. O hospedeiro A tem de enviar um pacote de L bits ao hospedeiro B.

a. Expresse o atraso de propagação, d_{prop}, em termos de m e s.

b. Determine o tempo de transmissão do pacote, d_{trans}, em termos de L e R.

c. Ignorando os atrasos de processamento e de fila, obtenha uma expressão para o atraso fim a fim.

d. Suponha que o hospedeiro A comece a transmitir o pacote no instante $t = 0$. No instante $t = d_{trans}$, onde estará o último bit do pacote?

e. Imagine que d_{prop} seja maior do que d_{trans}. Onde estará o primeiro bit do pacote no instante $t = d_{trans}$?

f. Considere que d_{prop} seja menor do que d_{trans}. Onde estará o primeiro bit do pacote no instante $t = d_{trans}$?

g. Suponha que $s = 2{,}5 \cdot 10^8$, $L = 120$ bits e $R = 56$ kbits/s. Encontre a distância m de modo que d_{prop} seja igual a d_{trans}.

P7. Neste problema, consideramos o envio de voz em tempo real do hospedeiro A para o hospedeiro B por meio de uma rede de comutação de pacotes (VoIP). O hospedeiro A converte voz analógica para uma cadeia digital de bits de 64 kbits/s e, em seguida, agrupa os bits em pacotes de 56 bytes. Há apenas um enlace entre os hospedeiros A e B; sua taxa de transmissão é de 2 Mbits/s e seu atraso de propagação, de 10 ms. Assim que o hospedeiro A recolhe um pacote, ele o envia ao hospedeiro B. Quando recebe um pacote completo, o hospedeiro B converte os bits do pacote em um sinal analógico. Quanto tempo decorre entre o momento em que um bit é criado (a partir do sinal analógico no hospedeiro A) e o momento em que ele é decodificado (como parte do sinal analógico no hospedeiro B)?

P8. Suponha que usuários compartilhem um enlace de 3 Mbits/s e que cada usuário precise de 150 kbits/s para transmitir, mas que transmita apenas durante 10% do tempo. (Veja a discussão sobre comutação de pacotes *versus* comutação de circuitos na Seção 1.3.)

a. Quando é utilizada comutação de circuitos, quantos usuários podem ser aceitos?

b. Para o restante deste problema, suponha que seja usada a comutação de pacotes. Determine a probabilidade de que determinado usuário esteja transmitindo.

c. Suponha que haja 120 usuários. Determine a probabilidade que, a um tempo dado, exatamente n usuários estejam transmitindo simultaneamente. (*Dica*: Use a distribuição binomial.)

d. Determine a probabilidade de haver 21 ou mais usuários transmitindo simultaneamente.

P9. Considere a discussão na Seção 1.3 sobre comutação de pacotes *versus* comutação de circuitos, na qual é dado um exemplo com um enlace de 1 Mbit/s. Quando em atividade, os usuários estão gerando dados a uma taxa de 100 kbits/s; mas a probabilidade de estarem em atividade, gerando dados, é de $p = 0{,}1$. Suponha que o enlace de 1 Mbit/s seja substituído por um de 1 Gbit/s.

a. Qual é o número máximo de usuários, N, que pode ser suportado simultaneamente por comutação de pacotes?

b. Agora considere comutação de circuitos e um número M de usuários. Elabore uma fórmula (em termos de p, M, N) para a probabilidade de que mais de N usuários estejam enviando dados.

P10. Considere um pacote de comprimento L que se inicia no sistema final A e percorre três enlaces até um sistema final de destino. Eles estão conectados por dois comutadores de pacotes. Suponha que d_i, s_i e R_i representem o comprimento, a velocidade de propagação e a taxa de transmissão do enlace i, sendo $i = 1$, 2, 3. O comutador de pacote atrasa cada pacote por d_{proc}. Considerando que não haja nenhum atraso de fila, em relação a d_i, s_i e R_i, ($i = 1, 2, 3$) e L, qual é o atraso fim a fim total para o pacote? Suponha agora que o pacote tenha 1.500 bytes, a velocidade de propagação de ambos os enlaces seja $2{,}5 \cdot 10^8$ m/s, as taxas de transmissão dos três enlaces sejam 2 Mbits/s, o atraso de processamento do comutador de pacotes seja de 3 ms, o comprimento do primeiro enlace seja 5.000 km, o do segundo seja 4.000 km e do último 1.000 km. Dados esses valores, qual é o atraso fim a fim?

P11. No problema anterior, suponha que $R_1 = R_2 = R_3 = R$ e $d_{proc} = 0$. Suponha que o comutador de pacote não armazene e reenvie pacotes, mas transmita imediatamente cada bit recebido antes de esperar o pacote chegar. Qual é o atraso fim a fim?

P12. Um comutador de pacotes recebe um pacote e determina o enlace de saída pelo qual deve ser enviado. Quando o pacote chega, outro já está sendo transmitido nesse enlace de saída e outros quatro já estão esperando para serem transmitidos. Os pacotes são transmitidos em ordem de chegada. Suponha que todos os pacotes tenham 1.500 bytes e que a taxa do enlace seja 2 Mbits/s. Qual é o atraso de fila para o pacote? De modo geral, qual é o atraso de fila quando todos os pacotes possuem comprimento L, a taxa de transmissão é R, x bits do pacote sendo transmitido já foram transmitidos e N pacotes já estão na fila?

P13. (a) Suponha que N pacotes cheguem simultaneamente ao enlace no qual não há pacotes sendo transmitidos e nem pacotes enfileirados. Cada pacote tem L de comprimento e é transmitido à taxa R. Qual é o atraso médio para os N pacotes?

(b) Agora considere que N desses pacotes cheguem ao enlace a cada LN/R segundos. Qual é o atraso de fila médio de um pacote?

P14. Considere o atraso de fila em um buffer de roteador, sendo I a intensidade de tráfego; isto é, $I = La/R$. Suponha que o atraso de fila tome a forma de $IL/R\ (1 - I)$ para $I < 1$.

a. Deduza uma fórmula para o atraso total, isto é, para o atraso de fila mais o atraso de transmissão.

b. Faça um gráfico do atraso total como uma função de L/R.

P15. Sendo a a taxa de pacotes que chegam a um enlace em pacotes/s, e μ a taxa de transmissão de enlaces em pacotes/s, baseado na fórmula do atraso total (isto é, o atraso de fila mais o atraso de transmissão) do problema anterior, deduza uma fórmula para o atraso total em relação a a e μ.

P16. Considere um buffer de roteador anterior a um enlace de saída. Neste problema, você usará a fórmula de Little, uma famosa fórmula da teoria das filas. Considere N o número médio de pacotes no buffer mais o pacote sendo transmitido, a a taxa de pacotes que chegam no enlace, e d o atraso total médio (isto é, o atraso de fila mais o atraso de transmissão) sofrido pelo pacote. Dada a fórmula de Little $N = a \cdot d$, suponha que, na média, o buffer contenha 10 pacotes, o atraso de fila de pacote médio seja 10 ms e a taxa de transmissão do enlace seja 100 pacotes/s. Utilizando tal fórmula, qual é a taxa média de chegada, considerando que não há perda de pacote?

P17. (a) Generalize a Equação 1.2 na Seção 1.4.3 para taxas de processamento heterogêneas, taxas de transmissão e atrasos de propagação.

(b) Repita o item (a), mas suponha também que haja um atraso definição fila médio d_{fila} em cada nó.

P18. Execute o programa Traceroute para verificar a rota entre uma origem e um destino, no mesmo continente, para três horários diferentes do dia.

a. Determine a média e o desvio-padrão dos atrasos de ida e volta para cada um dos três horários.

b. Determine o número de roteadores no caminho para cada um dos três. Os caminhos mudaram em algum dos horários?

c. Tente identificar o número de redes de ISP pelas quais o pacote do Traceroute passa entre origem e destino. Roteadores com nomes semelhantes e/ou endereços IP semelhantes devem ser considerados parte do mesmo ISP. Em suas respostas, os maiores atrasos ocorrem nas interfaces de formação de pares entre ISPs adjacentes?

d. Faça o mesmo para uma origem e um destino em continentes diferentes. Compare os resultados dentro do mesmo continente com os resultados entre continentes diferentes.

P19. (a) Visite o site <www.traceroute.org> e realize traceroutes de duas cidades diferentes na França para o mesmo hospedeiro de destino nos Estados Unidos. Quantos enlaces são iguais nos dois traceroutes? O enlace transatlântico é o mesmo?

(b) Repita (a), mas desta vez escolha uma cidade na França e outra cidade na Alemanha.

(c) Escolha uma cidade nos Estados Unidos e realize traceroutes para dois hosts, cada um em uma cidade diferente na China. Quantos enlaces são comuns nos dois traceroutes? Os dois traceroutes divergem antes de chegar à China?

P20. Considere o exemplo de vazão correspondente à Figura 1.20 (b). Agora imagine que haja M pares de cliente-servidor em vez de 10. R_s, R_c e R representam as taxas do enlace do servidor, enlaces do cliente e enlace da rede. Suponha que os outros enlaces possuam capacidade abundante e que não haja outro tráfego na rede além daquele gerado pelos M pares cliente-servidor. Deduza uma expressão geral para a vazão em relação a R_s, R_c, R e M.

P21. Considere a Figura 1.19(b). Agora suponha que haja M percursos entre o servidor e o cliente. Dois percursos nunca compartilham qualquer enlace. O percurso k ($k = 1, ..., M$) consiste em N enlaces com taxas de transmissão R_1^k, R_2^k...; R_N^K. Se o servidor pode usar somente um percurso para enviar dados ao cliente, qual é a vazão máxima que ele pode atingir? Se o servidor pode usar todos os M percursos para enviar dados, qual é a vazão máxima que ele pode atingir?

P22. Considere a Figura 1.19(b). Suponha que cada enlace entre o servidor e o cliente possua uma probabilidade de perda de pacote p, e que as probabilidades de perda de pacote para esses enlaces sejam independentes. Qual é a probabilidade de um pacote (enviado pelo servidor) ser recebido com sucesso pelo receptor? Se o pacote se perder no percurso do servidor para o cliente, então o servidor retransmitirá o pacote. Na média, quantas vezes o servidor retransmitirá o pacote para que o cliente o receba com sucesso?

P23. Considere a Figura 1.19(a). Suponha que o enlace de gargalo ao longo do percurso do servidor para o cliente seja o primeiro com a taxa R_s bits/s. Imagine que enviemos um par de pacotes um após o outro do servidor para o cliente, e que não haja outro tráfego nesse percurso. Suponha também que cada pacote de tamanho L bits e os dois enlaces tenham o mesmo atraso de propagação d_{prop}.

a. Qual é o tempo entre chegadas do pacote ao destino? Isto é, quanto tempo transcorre desde quando o último bit do primeiro pacote chega até quando o último bit do segundo pacote chega?

b. Agora suponha que o segundo enlace seja o de gargalo (isto é, $R_c < R_s$). É possível que o segundo pacote entre na fila de entrada do segundo enlace? Explique. Agora imagine que o servidor envie o segundo pacote T segundos após enviar o primeiro. Qual deverá ser o tamanho de T para garantir que não haja uma fila antes do segundo enlace? Explique.

P24. Imagine que você queira enviar, com urgência, 40 terabytes de dados de Boston para Los Angeles. Você tem disponível um enlace dedicado de 100 Mbits/s para transferência de dados. Escolheria transmitir os dados por meio desse enlace ou usar um serviço de entrega em 24 horas? Explique.

P25. Suponha que dois hospedeiros, A e B, estejam separados por uma distância de 20 mil quilômetros e conectados por um enlace direto de $R = 2$ Mbits/s. Suponha que a velocidade de propagação pelo enlace seja de $2{,}5 \cdot 10^8$ m/s.

a. Calcule o produto largura de banda-atraso $R \cdot d_{prop}$.

b. Considere o envio de um arquivo de 800 mil bits do hospedeiro A para o hospedeiro B. Suponha que o arquivo seja enviado continuamente, como se fosse uma única grande mensagem. Qual é o número máximo de bits que estará no enlace a qualquer dado instante?

c. Interprete o produto largura de banda × atraso.

d. Qual é o comprimento (em metros) de um bit no enlace? É maior do que o de um campo de futebol?

e. Derive uma expressão geral para o comprimento de um bit em termos da velocidade de propagação s, da velocidade de transmissão R e do comprimento do enlace m.

P26. Com referência ao Problema P25, suponha que possamos modificar R. Para qual valor de R o comprimento de um bit será o mesmo que o comprimento do enlace?

P27. Considere o Problema P25, mas agora com um enlace de $R = 1$ Gbit/s.

a. Calcule o produto largura de banda × atraso, $R \times d_{prop}$.

b. Considere o envio de um arquivo de 800 mil bits do hospedeiro A para o computador B. Suponha que o arquivo seja enviado continuamente, como se fosse uma única grande mensagem. Qual será o número máximo de bits que estará no enlace a qualquer dado instante?

c. Qual é o comprimento (em metros) de um bit no enlace?

P28. Novamente com referência ao Problema P25.

a. Quanto tempo demora para mandar o arquivo, admitindo que ele seja enviado continuamente?

b. Suponha agora que o arquivo seja fragmentado em 20 pacotes e que cada um contenha 40 mil bits. Imagine que cada pacote seja verificado pelo receptor e que o tempo de transmissão de uma verificação de pacote seja desprezível. Por fim, admita que o emissor não possa enviar um pacote até que o anterior tenha sido reconhecido. Quanto tempo demorará para enviar o arquivo?

c. Compare os resultados de 'a' e 'b'.

P29. Suponha que haja um enlace de micro-ondas de 10 Mbits/s entre um satélite geoestacionário e sua estação-base na Terra. A cada minuto o satélite tira uma foto digital e a envia à estação-base. Considere uma velocidade de propagação de $2{,}4 \cdot 10^8$ m/s.

a. Qual é o atraso de propagação do enlace?

b. Qual é o produto largura de banda-atraso, $R \cdot d_{prop}$?

c. Seja x o tamanho da foto. Qual é o valor mínimo de x para que o enlace de micro-ondas transmita continuamente?

P30. Considere a analogia da viagem aérea que utilizamos em nossa discussão sobre camadas na Seção 1.5, e o acréscimo de cabeçalhos a unidades de dados de protocolo enquanto passam pela pilha. Existe uma noção equivalente de acréscimo de informações de cabeçalho à movimentação de passageiros e suas malas pela pilha de protocolos da linha aérea?

P31. Em redes modernas de comutação de pacotes, inclusive a Internet, o hospedeiro de origem segmenta mensagens longas de camada de aplicação (por exemplo, uma imagem ou um arquivo de música) em pacotes menores e os envia pela rede. O destinatário, então, monta novamente os pacotes restaurando a mensagem original. Denominamos esse processo *segmentação de mensagem*. A Figura 1.27 ilustra o transporte fim a fim de uma mensagem com e sem segmentação. Considere que uma mensagem de 8×10^6 bits de comprimento tenha de ser enviada da origem ao destino na Figura 1.27. Suponha que a velocidade de cada enlace da figura seja 2 Mbits/s. Ignore atrasos de propagação, de fila e de processamento.

a. Considere o envio da mensagem da origem ao destino *sem* segmentação. Quanto tempo essa mensagem levará para ir do hospedeiro de origem até o primeiro comutador de pacotes? Tendo em mente que cada comutador usa comutação de pacotes do tipo armazena-e-reenvia, qual é o tempo total para levar a mensagem do hospedeiro de origem ao hospedeiro de destino?

b. Agora suponha que a mensagem seja segmentada em 800 pacotes, cada um com 10.000 bits de comprimento. Quanto tempo demorará para o primeiro pacote ir do hospedeiro de origem até o primeiro comutador?

FIGURA 1.27 TRANSPORTE FIM A FIM DE MENSAGEM: (A) SEM SEGMENTAÇÃO DE MENSAGEM; (B) COM SEGMENTAÇÃO DE MENSAGEM

Quando o primeiro pacote está sendo enviado do primeiro ao segundo comutador, o segundo pacote está sendo enviado da máquina de origem ao primeiro comutador. Em que instante o segundo pacote terá sido completamente recebido no primeiro comutador?

c. Quanto tempo demorará para movimentar o arquivo do hospedeiro de origem até o hospedeiro de destino quando é usada segmentação de mensagem? Compare este resultado com sua resposta no item 'a' e comente.

d. Além de reduzir o atraso, quais são as razões para usar a segmentação de mensagem?

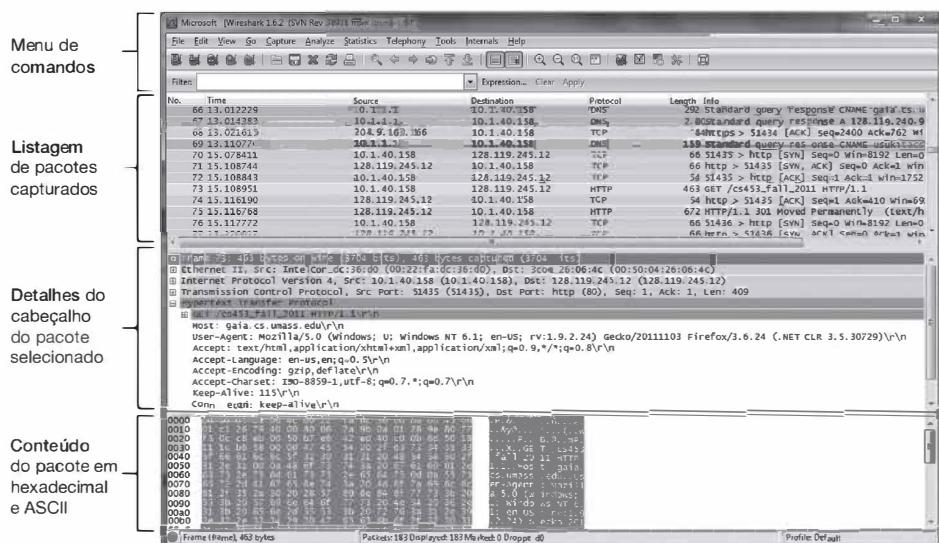
e. Discuta as desvantagens da segmentação de mensagem.

P32. Experimente o applet "Message Segmentation" apresentado no site deste livro. Os atrasos no applet correspondem aos atrasos obtidos no problema anterior? Como os atrasos de propagação no enlace afetam o atraso total fim a fim na comutação de pacotes (com segmentação de mensagem) e na comutação de mensagens?

P33. Considere o envio de um arquivo grande de F bits do hospedeiro A para o hospedeiro B. Há dois enlaces (e dois comutadores) entre A e B, e os enlaces não estão congestionados (isto é, não há atrasos de fila). O hospedeiro A fragmenta o arquivo em segmentos de S bits cada e adiciona 80 bits de cabeçalho a cada segmento, formando pacotes de $L = 80 + S$ bits. Cada enlace tem uma taxa de transmissão de R bits/s. Qual é o valor de S que minimiza o atraso para levar o arquivo de A para B? Desconsidere o atraso de propagação.

P34. O Skype oferece um serviço que lhe permite fazer uma ligação telefônica de um PC para um telefone comum. Isso significa que a chamada de voz precisa passar pela Internet e por uma rede telefônica. Discuta como isso poderia ser feito.

WIRESHARK LAB

"Conte-me e eu esquecerei. Mostre-me e eu lembrarei. Envolva-me e eu entenderei."
Provérbio chinês

A compreensão de protocolos de rede pode ser muito mais profunda se os virmos em ação e interagirmos com eles — observando a sequência de mensagens trocadas entre duas entidades de protocolo, pesquisando detalhes de sua operação, fazendo que eles executem determinadas ações e observando tais ações e suas consequências. Isso pode ser feito em cenários simulados ou em um ambiente real de rede, tal como a Internet. Os applets Java apresentados (em inglês) no site deste livro adotam a primeira abordagem. Nos Wireshark labs adotaremos a última. Você executará aplicações de rede em vários cenários utilizando seu computador no escritório, em casa

ou em um laboratório e observará também os protocolos de rede interagindo e trocando mensagens com entidades de protocolo que estão executando em outros lugares da Internet. Assim, você e seu computador serão partes integrantes desses laboratórios ao vivo. Você observará — e aprenderá — fazendo.

A ferramenta básica para observar as mensagens trocadas entre entidades de protocolos em execução é denominada **analisador de pacotes** (*packet sniffer*). Como o nome sugere, um analisador de pacotes copia (fareja) passivamente mensagens enviadas e recebidas por seu computador; também exibe o conteúdo dos vários campos de protocolo das mensagens que captura. Uma tela do analisador de pacotes Wireshark é mostrada na Figura 1.28. O Wireshark é um analisador de pacotes gratuito que funciona em computadores com sistemas operacionais Windows, Linux/Unix e Mac. Por todo o livro, você encontrará Wireshark labs que o habilitarão a explorar vários dos protocolos estudados em cada capítulo. Neste primeiro Wireshark lab, você obterá e instalará uma cópia do programa, acessará um site e examinará as mensagens de protocolo trocadas entre seu navegador e o servidor Web.

Você encontrará detalhes completos, em inglês, sobre este primeiro Wireshark Lab (incluindo instruções sobre como obter e instalar o programa) no site de apoio deste livro.

FIGURA 1.28 UMA AMOSTRA DE TELA DO PROGRAMA WIRESHARK (AMOSTRA DE TELA DO WIRESHARK REIMPRESSA COM PERMISSÃO DA WIRESHARK FOUNDATION)

ENTREVISTA

Leonard Kleinrock

Leonard Kleinrock é professor de ciência da computação da Universidade da Califórnia em Los Angeles. Em 1969, seu computador na UCLA se tornou o primeiro nó da Internet. Ele criou os princípios da comutação de pacotes em 1961, que se tornou a tecnologia básica da Internet. Ele é bacharel em engenharia elétrica pela City College of New York (CCNY) e mestre e doutor em engenharia elétrica pelo Instituto de Tecnologia de Massachusetts (MIT).

O que o fez se decidir pela especialização em tecnologia de redes/Internet?

Como doutorando do MIT em 1959, percebi que a maioria dos meus colegas de turma estava fazendo pesquisas nas áreas de teoria da informação e de teoria da codificação. Havia no MIT o grande pesquisador Claude Shannon, que já tinha proposto estudos nessas áreas e resolvido a maior parte dos problemas importantes. Os problemas que restaram para pesquisar eram difíceis e de menor importância. Portanto, decidi propor uma nova área na qual até então ninguém tinha pensado. Lembre-se de que no MIT eu estava cercado de computadores, e era evidente para mim que em breve aquelas máquinas teriam de se comunicar umas com as outras. Na época, não havia nenhum meio eficaz de fazer isso; portanto, decidi desenvolver a tecnologia que permitiria a criação de redes de dados eficientes e confiáveis.

Qual foi seu primeiro emprego no setor de computação? O que ele envolvia?

Frequentei o curso noturno de bacharelado em engenharia elétrica da CCNY de 1951 a 1957. Durante o dia, trabalhei de início como técnico e depois como engenheiro em uma pequena empresa de eletrônica industrial chamada Photobell. Enquanto trabalhava lá, introduzi tecnologia digital na linha de produtos da empresa. Basicamente, estávamos usando equipamentos fotoelétricos para detectar a presença de certos itens (caixas, pessoas etc.) e a utilização de um circuito conhecido na época como *multivibrador biestável* era exatamente o tipo de tecnologia de que precisávamos para levar o processamento digital a esse campo da detecção. Acontece que esses circuitos são os blocos de construção básicos dos computadores e vieram a ser conhecidos como *flip-flops* ou *chaves* na linguagem coloquial de hoje.

O que passou por sua cabeça quando enviou a primeira mensagem computador a computador (da UCLA para o Stanford Research Institute)?

Francamente, não fazíamos a menor ideia da importância daquele acontecimento. Não havíamos preparado uma mensagem de significância histórica, como muitos criadores do passado o fizeram (Samuel Morse com "Que obra fez Deus.", ou Alexandre Graham Bell, com "Watson, venha cá! Preciso de você.", ou Neil Armstrong com "Este é um pequeno passo para o homem, mas um grande salto para a humanidade."). Esses caras eram *inteligentes!* Eles entendiam de meios de comunicação e relações públicas. Nosso objetivo foi nos conectar ao computador do SRI. Então digitamos a letra "L", que foi aceita corretamente, digitamos a letra "o", que foi aceita, e depois digitamos a letra "g", o que fez o hospedeiro no SRI pifar! Então, nossa mensagem acabou sendo curta e, talvez, a mais profética de todas, ou seja, "Lo!", como em *"Lo and behold"* (*Pasmem!*).

Antes, naquele mesmo ano, fui citado em um comunicado de imprensa da UCLA por ter dito que, logo que a rede estivesse pronta e em funcionamento, seria possível ter acesso a outros computadores a partir de nossa casa e escritório tão facilmente quanto tínhamos acesso à eletricidade e ao telefone. Portanto, a visão que eu tinha da Internet naquela época era que ela seria onipresente, estaria sempre em funcionamento e sempre disponível, que qualquer pessoa que possuísse qualquer equipamento poderia se conectar com ela de qualquer lugar e que ela seria invisível. Mas jamais imaginei que minha mãe, aos 99 anos de idade, usaria a Internet — e ela de fato usou!

Em sua opinião, qual é o futuro das redes?

A parte fácil da visão é predizer a infraestrutura por si mesma. Eu antecipo que vemos uma implantação considerável de computação nômade, aparelhos móveis e espaços inteligentes. Realmente, a disponibilidade da computação portátil, de alto desempenho, acessível e leve, e dos aparelhos de comunicação (mais a onipresença da Internet) que permitiu que nos tornássemos nômades. A computação nômade refere-se à tecnologia que permite aos usuários finais, que viajam de um lugar para o outro, ganhar acesso aos serviços da Internet de maneira transparente, não importando para onde vão e qual aparelho possuem ou ganham acesso. O difícil é predizer as aplicações e serviços, que nos surpreenderam consistentemente de formas dramáticas (e-mail, tecnologias de busca, a World Wide Web, blogs, redes sociais, geração de usuários e compartilhamento de música, fotos, vídeos etc.). Estamos na margem de uma nova categoria de aplicações móveis, inovadoras e surpreendentes, presentes em nossos aparelhos portáteis.

O passo seguinte vai nos capacitar a sair do mundo misterioso do ciberespaço para o mundo físico dos espaços inteligentes. A tecnologia dará vida a nossos ambientes (mesas, paredes, veículos, relógios e cintos, entre outros) por meio de atuadores, sensores, lógica, processamento, armazenagem, câmeras, microfones, alto-falantes, monitores e comunicação. Essa tecnologia

embutida permitirá que nosso ambiente forneça os serviços IP que quisermos. Quando eu entrar em uma sala, ela saberá que entrei. Poderei me comunicar com meu ambiente naturalmente, como se estivesse falando o meu idioma nativo; minhas solicitações gerarão respostas apresentadas como páginas Web em painéis de parede, por meus óculos, por voz, por hologramas e assim por diante.

Analisando um panorama mais longínquo, vejo um futuro para as redes que inclui componentes fundamentais que ainda virão. Vejo agentes inteligentes de software distribuídos por toda a rede, cuja função é fazer mineração de dados, agir sobre esses dados, observar tendências e realizar tarefas de formas dinâmicas e adaptativas. Vejo tráfego de rede consideravelmente maior gerado não tanto por seres humanos, mas por esses equipamentos embutidos e agentes inteligentes de software. Vejo grandes conjuntos de sistemas auto-organizáveis controlando essa rede imensa e veloz. Vejo quantidades enormes de informações zunindo por essa rede instantaneamente e passando por processamento e filtragem extraordinários. A Internet será, basicamente, um sistema nervoso de presença global. Vejo tudo isso e mais enquanto entramos de cabeça no século XXI.

Que pessoas o inspiraram profissionalmente?

Sem dúvida alguma, quem mais me inspirou foi Claude Shannon, do MIT, um brilhante pesquisador que tinha a capacidade de relacionar suas ideias matemáticas com o mundo físico de modo muitíssimo intuitivo. Ele fazia parte da banca examinadora de minha tese de doutorado.

Você pode dar algum conselho aos estudantes que estão ingressando na área de redes/Internet?

A Internet, e tudo o que ela habilita, é uma vasta fronteira nova, cheia de desafios surpreendentes. Há espaço para grandes inovações. Não fiquem limitados à tecnologia existente hoje. Soltem sua imaginação, pensem no que poderia acontecer e transformem isso em realidade.

CAPÍTULO

2

CAMADA DE APLICAÇÃO

Aplicações de rede são a razão de ser de uma rede de computadores. Se não fosse possível inventar aplicações úteis, não haveria necessidade de projetar protocolos de rede para suportá-las. Desde o surgimento da Internet, foram criadas numerosas aplicações úteis e divertidas. Elas têm sido a força motriz por trás do sucesso da Internet, motivando pessoas em lares, escolas, governos e empresas a tornarem a rede uma parte integral de suas atividades diárias.

Entre as aplicações da Internet estão as aplicações clássicas de texto, que se tornaram populares nas décadas de 1970 e 1980: correio eletrônico, acesso a computadores remotos, transferência de arquivo e grupos de discussão. Também há uma aplicação que alcançou estrondoso sucesso em meados da década de 1990: a World Wide Web, abrangendo a navegação na Web, busca e o comércio eletrônico. Duas aplicações de enorme sucesso também surgiram no final do milênio — mensagem instantânea e compartilhamento de arquivos P2P. Desde 2000, temos visto uma explosão de aplicações populares de voz e vídeo, incluindo: voz sobre IP (VoIP) e videoconferência sobre IP, como Skype; distribuição de vídeo gerada pelo usuário, como YouTube; e filmes por demanda, como Netflix. Durante esse mesmo período, também vimos aparecerem jogos on-line com vários jogadores, bastante atraentes, como Second Life e World of Warcraft. Mais recentemente, vimos o surgimento de uma nova geração de aplicações de rede social, como Facebook e Twitter, que criaram redes humanas atraentes em cima da rede de roteadores e enlaces de comunicação da Internet. É evidente que não tem havido redução de aplicações novas e interessantes para Internet. Talvez alguns dos leitores deste texto criem a próxima geração de aplicações quentes para Internet!

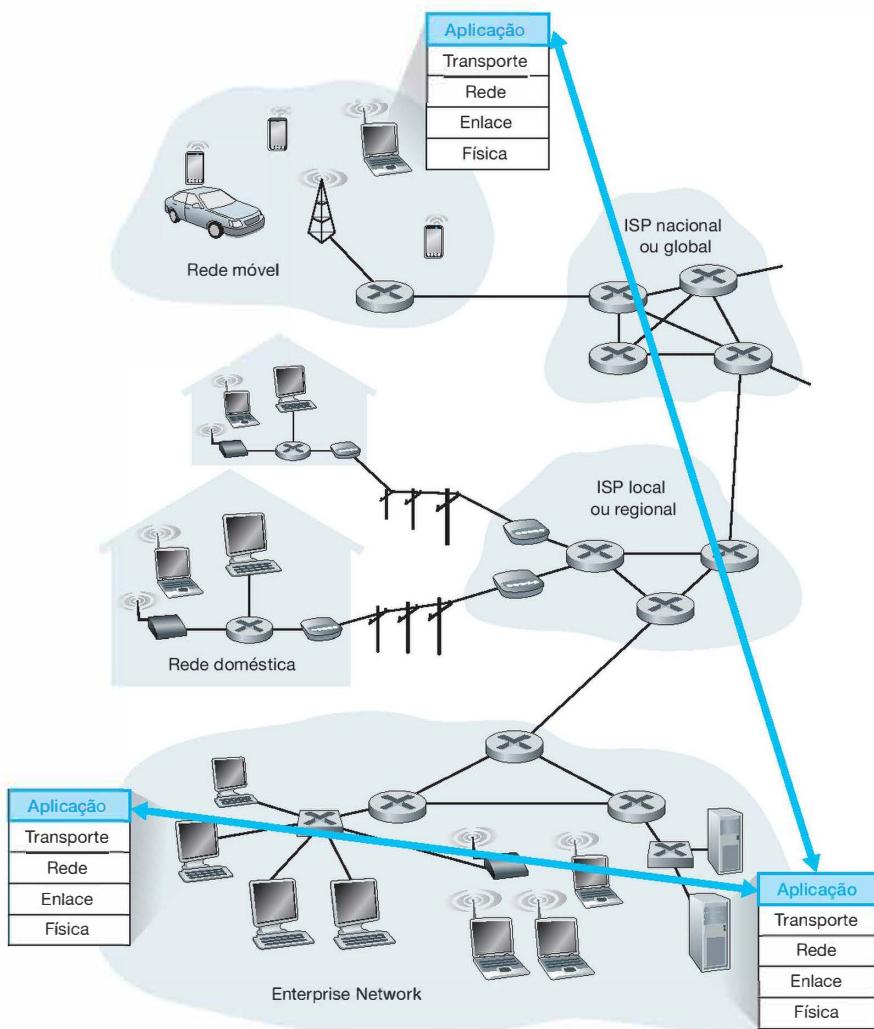
Neste capítulo estudaremos os aspectos conceituais e de implementação de aplicações de rede. Começaremos definindo conceitos fundamentais de camada de aplicação, incluindo serviços de rede exigidos por aplicações, clientes e servidores, processos e interfaces de camada de transporte. Vamos examinar detalhadamente várias aplicações de rede, entre elas a Web, e-mail, DNS e distribuição de arquivos P2P (o Capítulo 8 focaliza as aplicações multimídia, incluindo o vídeo por demanda e VoIP). Em seguida, abordaremos o desenvolvimento de aplicação de rede por TCP e também por UDP. Em particular, vamos estudar o API *socket* e examinar algumas aplicações cliente-servidor simples em Python. Apresentaremos também vários exercícios divertidos e interessantes de programação de aplicações no final do capítulo.

A camada de aplicação é um lugar particularmente bom para iniciarmos o estudo de protocolos. É terreno familiar, pois conhecemos muitas das aplicações que dependem dos que estudaremos. Ela nos dará uma boa ideia do que são protocolos e nos apresentará muitos assuntos que encontraremos de novo quando estudarmos protocolos de camadas de transporte, de rede e de enlace.

2.1 PRINCÍPIOS DE APLICAÇÕES DE REDE

Suponha que você tenha uma grande ideia para uma nova aplicação de rede. Essa aplicação será, talvez, um grande serviço para a humanidade, ou agradará a seu professor, ou fará de você uma pessoa rica; ou apenas será divertido desenvolvê-la. Seja qual for sua motivação, vamos examinar agora como transformar a ideia em uma aplicação do mundo real.

O núcleo do desenvolvimento de aplicação de rede é escrever programas que rodem em sistemas finais diferentes e se comuniquem entre si. Por exemplo, na aplicação Web há dois programas distintos que se comunicam um com o outro: o do navegador, que roda no hospedeiro do usuário (computador de mesa, laptop, tablet, smartphone e assim por diante); e o do servidor Web, que roda na máquina deste. Outro exemplo é um sistema de compartilhamento de arquivos P2P no qual há um programa em cada máquina que participa da comunidade de compartilhamento de arquivos. Nesse caso, os programas de cada máquina podem ser semelhantes ou idênticos.

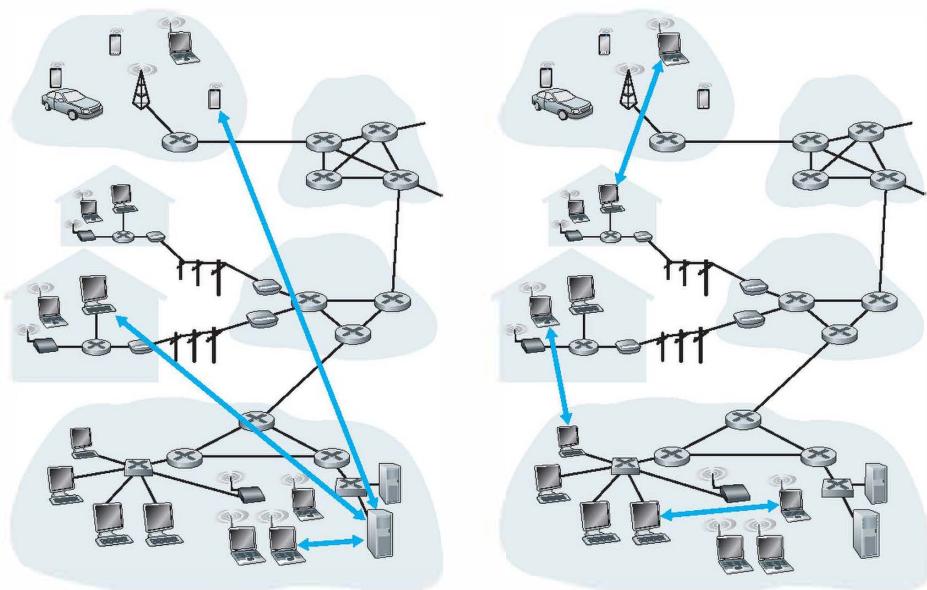
Portanto, ao desenvolver sua nova aplicação, você precisará escrever um software que rode em vários sistemas finais. Esse software poderia ser criado, por exemplo, em C, Java ou Python. Importante: você não precisará escrever programas que executem nos elementos do núcleo de rede, como roteadores e comutadores. Mesmo se quisesse, não poderia desenvolver programas para esses elementos. Como aprendemos no Capítulo 1 e mostramos na Figura 1.24, equipamentos de núcleo de rede não funcionam na camada de aplicação, mas em camadas mais baixas, em especial na de rede e abaixo dela. Esse projeto básico — a saber, confinar o software de aplicação nos sistemas finais —, como mostra a Figura 2.1, facilitou o desenvolvimento e a proliferação rápidos de uma vasta gama de aplicações de rede.

2.1.1 Arquiteturas de aplicação de rede

Antes de mergulhar na codificação do software, você deverá elaborar um plano geral para a arquitetura da sua aplicação. Tenha sempre em mente que a arquitetura de uma aplicação é bastante diferente da arquitetura da rede (por exemplo, a arquitetura em cinco camadas da Internet que discutimos no Capítulo 1). Do ponto de vista do profissional que desenvolve a aplicação, a arquitetura de rede é fixa e provê um conjunto específico de serviços. Por outro lado, a **arquitetura da aplicação** é projetada pelo programador e determina como a aplicação é organizada nos vários sistemas finais. Ao escolher a arquitetura da aplicação, é provável que o programador aproveite uma das duas arquiteturas mais utilizadas em aplicações modernas de rede: cliente-servidor ou P2P.

Em uma **arquitetura cliente-servidor** há um hospedeiro sempre em funcionamento, denominado *servidor*, que atende a requisições de muitos outros hospedeiros, denominados *clientes*. Um exemplo clássico é a aplicação Web na qual um servidor Web que está sempre em funcionamento atende a solicitações de navegadores de hospedeiros clientes. Quando recebe uma requisição de um objeto de um hospedeiro cliente, um servidor Web responde enviando o objeto solicitado. Observe que, na arquitetura cliente-servidor, os clientes não se comunicam diretamente uns com os outros; por exemplo, na aplicação Web, dois navegadores não se comunicam de modo direto. Outra característica dessa arquitetura é que o servidor tem um endereço fixo, bem conhecido, denominado endereço IP (que discutiremos em breve). Por causa dessa característica do servidor e pelo fato de ele estar sempre em funcionamento, um cliente sempre pode contatá-lo, enviando um pacote ao endereço do servidor. Algumas das aplicações mais conhecidas que empregam a arquitetura cliente-servidor são Web, FTP, Telnet e e-mail. Essa arquitetura é mostrada na Figura 2.2(a).

Em aplicações cliente-servidor, muitas vezes acontece de um único hospedeiro servidor ser incapaz de atender a todas as requisições de seus clientes. Por exemplo, um site popular de redes sociais pode ficar logo saturado se tiver apenas um servidor para atender a todas as solicitações. Por essa razão, um **datacenter**, acomodando um grande número de hospedeiros, é usado com frequência para criar um servidor virtual poderoso. Os serviços de Internet mais populares — como mecanismos de busca (por exemplo, Google e Bing), comércio via Internet (por exemplo, Amazon e eBay), e-mail baseado na Web (por exemplo, Gmail e Yahoo Mail), rede social (por exemplo, Facebook e Twitter) — empregam um ou mais centros de dados. Conforme discutimos

FIGURA 2.1 A COMUNICAÇÃO DE UMA APLICAÇÃO DE REDE OCORRE ENTRE SISTEMAS FINAIS NA CAMADA DE APLICAÇÃO

na Seção 1.3.3, o Google tem de 30 a 50 datacenters distribuídos no mundo inteiro, que em conjunto tratam de busca, YouTube, Gmail e outros serviços. Um datacenter pode ter centenas de milhares de servidores, que precisam ser alimentados e mantidos. Além disso, os provedores de serviços têm de pagar pelos custos de interconexão recorrente e largura de banda para o envio de dados a partir de seus datacenters.

Em uma **arquitetura P2P**, há uma confiança mínima (ou nenhuma) nos servidores dedicados nos centros de dados. Em vez disso, a aplicação utiliza a comunicação direta entre duplas de hospedeiros conectados alternadamente, denominados *pares*. Eles não são de propriedade dos provedores de serviço, mas são controlados por usuários de computadores de mesa e laptops, cuja maioria se aloja em residências, universidades e escritórios. Como os pares se comunicam sem passar por nenhum servidor dedicado, a arquitetura é denominada par a par (*peer-to-peer* — P2P). Muitas das aplicações de hoje mais populares e de intenso tráfego são

baseadas nas arquiteturas P2P, incluindo compartilhamento de arquivos (por exemplo, BitTorrent), aceleração de *download* assistida por par (por exemplo, Xunlei), telefonia por Internet (por exemplo, Skype) e IPTV (por exemplo, KanKan e PPstream). Essa arquitetura está ilustrada na Figura 2.2(b). Mencionamos que algumas aplicações possuem arquiteturas híbridas, combinando elementos cliente-servidor e P2P. Para muitas aplicações de mensagem instantânea, os servidores costumam rastrear o endereço IP dos usuários, mas as mensagens entre usuários são enviadas diretamente entre os hospedeiros do usuário (sem passar por servidores intermediários).

Uma das características mais fortes da arquitetura P2P é sua **autoescalabilidade**. Por exemplo, em uma aplicação de compartilhamento de arquivos P2P, embora cada par gere uma carga de trabalho solicitando arquivos, também acrescenta capacidade de serviço ao sistema distribuindo arquivos a outros pares. As arquiteturas P2P também possuem uma boa relação custo-benefício, visto que em geral não requerem infraestrutura e largura de banda de servidor significativas (ao contrário de projetos cliente-servidor com centros de dados). Entretanto, as futuras aplicações P2P estão diante de três principais desafios:

1. *ISP Amigável.* A maioria dos ISPs residenciais (incluindo o DSL e os ISPs a cabo) foi dimensionada para uso de largura de banda "assimétrica", ou seja, para muito mais tráfego de entrada do que de saída. Mas a transmissão de vídeo P2P e as aplicações de distribuição de vídeo transferem o tráfego de saída dos servidores para ISPs residenciais, colocando, assim, uma pressão significativa nos ISPs. As futuras aplicações P2P precisam ser criadas para que sejam amigáveis aos ISPs [Xie, 2008].
2. *Segurança.* Em razão de sua natureza altamente distribuída e exposta, as aplicações P2P podem ser um desafio para proteger [Doucer, 2002; Yu, 2006; Liang, 2006; Naoumov, 2006; Dhungel, 2008; LeBlond 2011].
3. *Incentivos.* O sucesso das futuras aplicações P2P também depende de usuários participativos para oferecer largura de banda, armazenamento e recursos da computação às aplicações, um projeto desafiador de incentivo [Feldman, 2005; Piatek, 2008; Aperjis, 2008; Liu, 2010].

FIGURA 2.2 (A) ARQUITETURA CLIENTE-SERVIDOR; (B) ARQUITETURA P2P

a. Arquitetura cliente-servidor

b. Arquitetura P2P

2.1.2 Comunicação entre processos

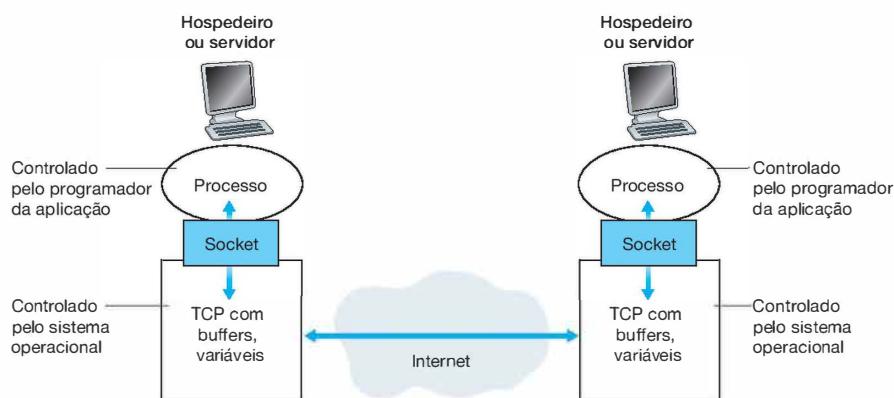
Antes de construir sua aplicação de rede, você também precisará ter um entendimento básico de como programas que rodam em vários sistemas finais comunicam-se entre si. No jargão de sistemas operacionais, na verdade não são programas, mas **processos** que se comunicam. Um processo pode ser imaginado como um programa que está rodando dentro de um sistema final. Quando os processos rodam no mesmo sistema final, comunicam-se usando comunicação interprocessos, cujas regras são determinadas pelo sistema operacional do sistema final. Porém, neste livro, não estamos interessados na comunicação entre processos do mesmo hospedeiro, mas em como se comunicam os que rodam em sistemas finais *diferentes* (com sistemas operacionais potencialmente diferentes).

Os processos em dois sistemas finais diferentes se comunicam trocando **mensagens** por meio da rede de computadores. Um processo originador cria e envia mensagens para a rede; um processo destinatário recebe-as e responde, devolvendo outras. A Figura 2.1 mostra que processos se comunicam usando a camada de aplicação da pilha de cinco camadas da arquitetura.

Processos clientes e processos servidores

Uma aplicação de rede consiste em pares de processos que enviam mensagens uns para os outros por meio de uma rede. Por exemplo, na aplicação Web, o processo navegador de um cliente troca mensagens com o de um servidor Web. Em um sistema de compartilhamento de arquivos P2P, um arquivo é transferido de um processo que está em um par para um que está em outro par. Para cada par de processos comunicantes normalmente rotulamos um dos dois processos de **cliente** e o outro, de **servidor**. Na Web, um navegador é um processo cliente e um servidor Web é um processo servidor. No compartilhamento de arquivos P2P, o par que envia o arquivo é rotulado de cliente e o que recebe, de servidor.

Talvez você já tenha observado que, em algumas aplicações, tal como compartilhamento de arquivos P2P, um processo pode ser ambos, cliente e servidor. De fato, um processo em um sistema de compartilhamento de arquivos P2P pode carregar e descarregar arquivos. Mesmo assim, no contexto de qualquer dada sessão entre um par de processos, ainda podemos rotular um processo de cliente e o outro de servidor. Definimos os processos cliente e servidor como segue:

> *No contexto de uma sessão de comunicação entre um par de processos, aquele que inicia a comunicação (isto é, o primeiro a contatar o outro no início da sessão) é rotulado de* ***cliente****. O que espera ser contatado para iniciar a sessão é o* ***servidor****.*

Na Web, um processo do navegador inicia o contato com um processo do servidor Web; por conseguinte, o processo do navegador é o cliente e o do servidor Web é o servidor. No compartilhamento de arquivos P2P, quando o Par A solicita ao Par B o envio de um arquivo específico, o Par A é o cliente enquanto o Par B é o servidor no contexto dessa sessão de comunicação. Quando não houver possibilidade de confusão, às vezes usaremos também a terminologia "lado cliente e lado servidor de uma aplicação". No final deste capítulo examinaremos passo a passo um código simples para ambos os lados de aplicações de rede: o lado cliente e o lado servidor.

A interface entre o processo e a rede de computadores

Como dissemos anteriormente, a maioria das aplicações consiste em pares de processos comunicantes, e os dois processos de cada par enviam mensagens um para o outro. Qualquer mensagem enviada de um processo para outro tem de passar pela rede subjacente. Um processo envia mensagens para a rede e recebe mensagens dela através de uma interface de software denominada ***socket***. Vamos considerar uma analogia que nos auxiliará a entender processos e *sockets*. Um processo é semelhante a uma casa e seu *socket*, à porta da casa. Quando um processo quer enviar uma mensagem a outro processo em outro hospedeiro, ele empurra a mensagem pela porta (*socket*). O emissor admite que exista uma infraestrutura de transporte do outro lado de sua porta que transportará a mensagem pela rede até a porta do processo destinatário. Ao chegar ao hospedeiro destinatário, a mensagem passa pela porta (*socket*) do processo receptor, que então executa alguma ação sobre a mensagem.

A Figura 2.3 ilustra a comunicação por *socket* entre dois processos que se comunicam pela Internet. (A Figura 2.3 admite que o protocolo de transporte subjacente usado pelos processos é o TCP.) Como mostra essa figura, um *socket* é a interface entre a camada de aplicação e a de transporte dentro de um hospedeiro. É também denominado **interface de programação da aplicação** (*application programming interface* — API) entre a aplicação e a rede, visto que é a interface de programação pela qual as aplicações de rede são criadas. O programador da aplicação controla tudo o que existe no lado da camada de aplicação do *socket*, mas tem pouco controle do lado de sua camada de transporte. Os únicos controles que o programador da aplicação tem do lado da camada de transporte são: (1) a escolha do protocolo de transporte e (2), talvez, a capacidade de determinar alguns parâmetros, tais como tamanho máximo de buffer e de segmentos (a serem abordados no Capítulo 3). Uma vez escolhido um protocolo de transporte, (se houver escolha) o programador constrói a aplicação usando os serviços da camada de transporte oferecidos por esse protocolo. Examinaremos *sockets* mais detalhadamente na Seção 2.7.

FIGURA 2.3 PROCESSOS DE APLICAÇÃO, *SOCKETS* E PROTOCOLO DE TRANSPORTE SUBJACENTE

Endereçando processos

Para enviar correspondência postal a determinado destino, este precisa ter um endereço. De modo semelhante, para que um processo rodando em um hospedeiro envie pacotes a um processo rodando em outro hospedeiro, o receptor precisa ter um endereço. Para identificar o processo receptor, duas informações devem ser especificadas: (1) o endereço do hospedeiro e (2) um identificador que especifica o processo receptor no hospedeiro de destino.

Na Internet, o hospedeiro é identificado por seu **endereço IP**. Discutiremos os endereços IP com mais detalhes no Capítulo 4. Por enquanto, tudo o que precisamos saber é que um endereço IP é uma quantidade de 32 bits que podemos pensar como identificando um hospedeiro de forma exclusiva. Além de saber o endereço do hospedeiro ao qual a mensagem é destinada, o processo de envio também precisa identificar o processo receptor (mais especificamente, o *socket* receptor) executando no hospedeiro. Essa informação é necessária porque, em geral, um hospedeiro poderia estar executando muitas aplicações de rede. Um **número de porta** de destino atende a essa finalidade. Aplicações populares receberam números de porta específicos. Por exemplo, um servidor Web é identificado pelo número de porta 80. Um processo servidor de correio (usando o protocolo SMTP) é identificado pelo número de porta 25. Uma lista dos números de porta conhecidos para todos os protocolos-padrão da Internet poderá ser encontrada em <http://www.iana.org>. Vamos examinar os números de porta em detalhes no Capítulo 3.

2.1.3 Serviços de transporte disponíveis para aplicações

Lembre-se de que um *socket* é a interface entre o processo da aplicação e o protocolo de camada de transporte. A aplicação do lado remetente envia mensagens por meio do *socket*. Do outro lado, o protocolo

de camada de transporte tem a responsabilidade de levar as mensagens pela rede até o *socket* do processo destinatário.

Muitas redes, inclusive a Internet, oferecem mais de um protocolo de camada de transporte. Ao desenvolver uma aplicação, você deve escolher um dos protocolos de camada de transporte disponíveis. Como fazer essa escolha? O mais provável é que você avalie os serviços e escolha o protocolo que melhor atenda às necessidades de sua aplicação. A situação é semelhante a decidir entre ônibus ou avião como meio de transporte entre duas cidades. Você tem de optar, e cada modalidade de transporte oferece serviços diferentes. (Por exemplo, o ônibus oferece a facilidade da partida e da chegada no centro da cidade, ao passo que o avião tem menor tempo de viagem.)

Quais são os serviços que um protocolo da camada de transporte pode oferecer às aplicações que o chamem? Podemos classificar, de maneira geral, os possíveis serviços segundo quatro dimensões: transferência confiável de dados, vazão, temporização e segurança.

Transferência confiável de dados

Como discutido no Capítulo 1, os pacotes podem se perder dentro de uma rede de computadores. Um pacote pode, por exemplo, esgotar um buffer em um roteador, ou ser descartado por um hospedeiro ou um roteador após alguns de seus bits terem sido corrompidos. Para muitas aplicações — como correio eletrônico, transferência de arquivo, acesso a hospedeiro remoto, transferências de documentos da Web e aplicações financeiras — a perda de dados pode ter consequências devastadoras (no último caso, para o banco e para o cliente!). Assim, para suportar essas aplicações, algo deve ser feito para garantir que os dados enviados por uma extremidade da aplicação sejam transmitidos correta e completamente para a outra ponta. Se um protocolo fornecer um serviço de recebimento de dados garantido, ele fornecerá uma **transferência confiável de dados**. Um importante serviço que o protocolo da camada de transporte pode oferecer para uma aplicação é a transferência confiável de dados processo a processo. Quando um protocolo de transporte oferece esse serviço, o processo remetente pode apenas passar seus dados para um *socket* e saber com absoluta confiança que eles chegarão sem erro ao processo destinatário.

Quando um protocolo da camada de transporte não oferece uma transferência confiável de dados, os dados enviados pelo remetente talvez nunca cheguem ao destinatário. Isso pode ser aceitável para **aplicações tolerantes a perda**, em especial as de multimídia como áudio/vídeo em tempo real ou áudio/vídeo armazenado, que podem tolerar alguma perda de dados. Nessas aplicações, dados perdidos podem resultar em uma pequena falha durante a execução do áudio/vídeo — o que não é um prejuízo crucial.

Vazão

No Capítulo 1 apresentamos o conceito de vazão disponível, que, no contexto de sessão da comunicação entre dois processos ao longo de um caminho de rede, é a taxa pela qual o processo remetente pode enviar bits ao processo destinatário. Como outras sessões compartilharão a largura de banda no caminho da rede e estão indo e voltando, a vazão disponível pode oscilar com o tempo. Essas observações levam a outro serviço natural que um protocolo da camada de transporte pode oferecer, ou seja, uma vazão disponível garantida a uma taxa específica. Com tal serviço, a aplicação pode solicitar uma vazão garantida de *r* bits/s, e o protocolo de transporte garante, então, que a vazão disponível seja sempre *r* bits/s, pelo menos. Tal serviço de vazão garantida seria atraente para muitas aplicações. Por exemplo, se uma aplicação de telefonia por Internet codifica voz a 32 kbits/s, ela precisa enviar dados para a rede e fazer que sejam entregues na aplicação receptora à mesma taxa. Se o protocolo de transporte não puder fornecer essa vazão, a aplicação precisará codificar a uma taxa menor (e receber vazão suficiente para sustentar essa taxa de codificação mais baixa) ou então desistir, já que receber, digamos, metade da vazão de que precisa de nada ou pouco adianta para essa aplicação de telefonia por Internet. Aplicações que possuam necessidade de vazão são conhecidas como **aplicações sensíveis à largura de banda**. Muitas aplicações de multimídia existentes são sensíveis à largura de banda, embora algumas possam usar técnicas adaptativas para codificar a uma taxa que corresponda à vazão disponível na ocasião.

Embora aplicações sensíveis à largura de banda possuam necessidades específicas de vazão, **aplicações elásticas** podem usar qualquer quantidade mínima ou máxima que por acaso esteja disponível. Correio eletrônico, transferência de arquivos e transferências Web são todas aplicações elásticas. Claro, quanto mais vazão, melhor. Há um ditado que diz que "dinheiro nunca é demais"; nesse caso, podemos dizer que vazão nunca é demais!

Temporização

Um protocolo da camada de transporte pode também oferecer garantias de temporização. Como nas garantias de vazão, as de temporização podem surgir em diversos aspectos e modos. Citamos como exemplo o fato de que cada bit que o remetente insere no *socket* chega ao *socket* destinatário em menos de 100 ms depois. Esse serviço seria atrativo para aplicações interativas em tempo real, como a telefonia por Internet, ambientes virtuais, teleconferência e jogos multijogadores, que exigem restrições de temporização no envio de dados para garantir eficácia. (Veja Capítulo 7, [Gauthier, 1999; Ramjee, 1994].) Longos atrasos na telefonia por Internet, por exemplo, tendem a resultar em pausas artificiais na conversação; em um jogo multiusuário ou ambiente virtual interativo, um longo atraso entre realizar uma ação e ver a reação do ambiente (por exemplo, a reação de outro jogador na outra extremidade de uma conexão fim a fim) faz a aplicação parecer menos realista. Para aplicações que não são em tempo real, é sempre preferível um atraso menor a um maior, mas não há nenhuma limitação estrita aos atrasos fim a fim.

Segurança

Por fim, um protocolo de transporte pode oferecer um ou mais serviços de segurança a uma aplicação. Por exemplo, no hospedeiro remetente, um protocolo de transporte é capaz de codificar todos os dados transmitidos pelo processo remetente e, no hospedeiro destinatário, o protocolo da camada de transporte pode codificar os dados antes de enviá-los ao destinatário. Tal serviço pode oferecer sigilo entre os dois, mesmo que os dados sejam, de algum modo, observados entre os processos remetente e destinatário. Um protocolo de transporte consegue, além do sigilo, fornecer outros serviços de segurança, incluindo integridade dos dados e autenticação do ponto terminal, assuntos que serão abordados em detalhes no Capítulo 8.

2.1.4 Serviços de transporte providos pela Internet

Até aqui, consideramos serviços de transportes que uma rede de computadores *poderia* oferecer em geral. Vamos agora nos aprofundar mais no assunto e analisar o tipo de suporte de aplicação provido pela Internet. A Internet (e, em um amplo sentido, as redes TCP/IP) disponibiliza dois protocolos de transporte para aplicações, o UDP e o TCP. Quando você (como um criador de aplicação) cria uma nova aplicação de rede para a Internet, uma das primeiras decisões a ser tomada é usar o UDP ou o TCP. Cada um deles oferece um conjunto diferente de serviços para as aplicações solicitantes. A Figura 2.4 mostra os requisitos do serviço para algumas aplicações.

FIGURA 2.4 REQUISITOS DE APLICAÇÕES DE REDE SELECIONADAS

Aplicação	Perda de dados	Vazão	Sensibilidade ao tempo
Transferência / download de arquivo	Sem perda	Elástica	Não
E-mail	Sem perda	Elástica	Não
Documentos Web	Sem perda	Elástica (alguns kbits/s)	Não
Telefonia via Internet/ videoconferência	Tolerante à perda	Áudio: alguns kbits/s – 1Mbit/s Vídeo: 10 kbits/s – 5 Mbits/s	Sim: décimos de segundo
Áudio/vídeo armazenado	Tolerante à perda	Igual acima	Sim: alguns segundos
Jogos interativos	Tolerante à perda	Poucos kbits/s – 10 kbits/s	Sim: décimos de segundo
Mensagem instantânea	Sem perda	Elástico	Sim e não

Serviços do TCP

O modelo de serviço TCP inclui um serviço orientado para conexão e um serviço confiável de transferência de dados. Quando uma aplicação solicita o TCP como seu protocolo de transporte, recebe dele ambos os serviços.

- *Serviço orientado para conexão.* O TCP faz o cliente e o servidor trocarem informações de controle de camada de transporte *antes* que as mensagens de camada de aplicação comecem a fluir. Esse procedimento de apresentação, por assim dizer, alerta o cliente e o servidor, permitindo que eles se preparem para uma enxurrada de pacotes. Após a fase de apresentação, dizemos que existe uma **conexão TCP** entre os *sockets* dos dois processos. A conexão é *full-duplex* (simultânea), visto que os dois processos podem enviar mensagens um ao outro pela conexão ao mesmo tempo. Quando termina de enviar mensagens, a aplicação deve interromper a conexão. No Capítulo 3, discutiremos em detalhes serviço orientado para conexão e examinaremos como ele é implementado.
- *Serviço confiável de transporte.* Os processos comunicantes podem contar com o TCP para a entrega de todos os dados enviados sem erro e na ordem correta. Quando um lado da aplicação passa uma cadeia de bytes para dentro de um *socket*, pode contar com o TCP para entregar a mesma cadeia de dados ao *socket* receptor, sem falta de bytes nem bytes duplicados.

O TCP também inclui um mecanismo de controle de congestionamento, um serviço voltado ao bem-estar geral da Internet e não ao benefício direto dos processos comunicantes. O mecanismo de controle de congestionamento do TCP limita a capacidade de transmissão de um processo (cliente ou servidor) quando a rede está congestionada entre remetente e destinatário. Como veremos no Capítulo 3, o controle de congestionamento do TCP tenta limitar cada conexão do TCP à sua justa porção de largura de banda de rede.

Serviços do UDP

O UDP é um protocolo de transporte simplificado, leve, com um modelo de serviço minimalista. É um serviço não orientado para conexão; portanto, não há apresentação antes que os dois processos comecem a se comunicar. O UDP provê um serviço não confiável de transferência de dados — isto é, quando um processo envia uma mensagem para dentro de um *socket* UDP, o protocolo *não* oferece garantias de que a mensagem chegará ao processo receptor. Além do mais, mensagens que chegam de fato ao processo receptor podem chegar fora de ordem.

O UDP não inclui um mecanismo de controle de congestionamento; portanto, um processo originador pode bombear dados para dentro de uma camada abaixo (a de rede) à taxa que quiser. (Observe, entretanto, que a vazão fim a fim real pode ser menor do que essa taxa por causa da capacidade de transmissão limitada de enlaces intervenientes ou pelo congestionamento.)

Serviços não providos pelos protocolos de transporte da Internet

Organizamos os serviços do protocolo de transporte em quatro dimensões: transferência confiável de dados, vazão, temporização e segurança. Quais deles são providos pelo TCP e pelo UDP? Já vimos que o TCP fornece a transferência confiável de dados fim a fim, e sabemos também que ele pode ser aprimorado com facilidade na camada de aplicação com o SSL para oferecer serviços de segurança. Mas em nossa breve descrição sobre o TCP e o UDP faltou mencionar as garantias de vazão e de temporização — serviços *não* fornecidos pelos protocolos de transporte da Internet de hoje. Isso significa que as aplicações sensíveis ao tempo, como a telefonia por Internet, não podem rodar na rede atual? A resposta decerto é negativa — a Internet tem recebido essas aplicações por muitos anos. Tais aplicações muitas vezes funcionam bem, por terem sido desenvolvidas para lidar, na medida do possível, com a falta de garantia. Analisaremos vários desses truques de projeto no Capítulo 7. No entanto, o projeto inteligente possui suas limitações quando o atraso é excessivo, ou quando a vazão fim a fim é limitada.

SEGURANÇA EM FOCO

Protegendo o TCP

Nem o TCP ou o UDP fornecem qualquer codificação — os dados que o processo remetente transfere para seu *socket* são os mesmos que percorrem a rede até o processo destinatário. Então, por exemplo, se o processo destinatário enviar uma senha em texto claro (ou seja, não codificado) para seu *socket*, ela percorrerá por todos os enlaces entre o remetente e o destinatário, sendo analisada e descoberta em qualquer um dos enlaces intervenientes. Em razão de a privacidade e outras questões de segurança terem se tornado importantes para muitas aplicações, a comunidade da Internet desenvolveu um aperfeiçoamento para o TCP, denominado Camada de *Socket* Seguros (Secure Sockets Layer — SSL). O aperfeiçoamento SSL para o TCP não só faz tudo o que o TCP tradicional faz, como também oferece serviços importantes de segurança processo a processo, incluindo codificação, integridade dos dados e autenticação do ponto de chegada. Enfatizamos que o SSL não é um terceiro protocolo da Internet, no mesmo nível do TCP e do UDP, mas um aperfeiçoamento do TCP executado na camada de aplicação. Em particular, se uma aplicação quiser utilizar o serviço do SSL, é preciso incluir o código SSL (disponível na forma de classes e bibliotecas altamente otimizadas) da aplicação em ambas as partes cliente e servidor. O SSL possui sua própria API de *socket* que é semelhante à tradicional API de *socket* TCP. Quando uma aplicação utiliza o SSL, o processo remetente transfere dados em texto claro para o *socket* SSL; no hospedeiro emissor, então, o SSL codifica os dados e os passa para o *socket* TCP. Os dados codificados percorrem a Internet até o *socket* TCP no processo destinatário. O *socket* destinatário passa os dados codificados ao SSL, que os decodifica. Por fim, o SSL passa os dados em texto claro por seu *socket* SSL até o processo destinatário. Abordaremos o SSL em mais detalhes no Capítulo 8.

Em resumo, a Internet hoje pode oferecer serviços satisfatórios a aplicações sensíveis ao tempo, mas não garantias de temporização ou de largura de banda.

A Figura 2.5 mostra os protocolos de transporte usados por algumas aplicações populares da Internet. Vemos que e-mail, acesso a terminais remotos, a Web e transferência de arquivos usam o TCP. Essas aplicações escolheram o TCP principalmente porque ele oferece um serviço confiável de transferência de dados, garantindo que todos eles mais cedo ou mais tarde cheguem a seu destino. Como as aplicações de telefonia por Internet (como Skype) muitas vezes toleram alguma perda, mas exigem uma taxa mínima para que sejam eficazes, seus programadores em geral preferem rodá-las em cima do UDP, contornando assim o mecanismo de controle de congestionamento do TCP e evitando pacotes extras. Porém, como muitos *firewalls* são configurados para bloquear (quase todo) o tráfego UDP, as aplicações de telefonia por Internet quase sempre são projetadas para usar TCP como um apoio se a comunicação por UDP falhar.

FIGURA 2.5 APLICAÇÕES POPULARES DA INTERNET, SEUS PROTOCOLOS DE CAMADA DE APLICAÇÃO E SEUS PROTOCOLOS DE TRANSPORTE SUBJACENTES

Aplicação	Protocolo de camada de aplicação	Protocolo de transporte subjacente
Correio eletrônico	SMTP [RFC 5321]	TCP
Acesso a terminal remoto	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
Transferência de arquivos	FTP [RFC 959]	TCP
Multimídia em fluxo contínuo	HTTP (por exemplo, YouTube)	TCP
Telefonia por Internet	SIP [RFC 3261], RTP [RFC 3550] ou proprietária (por exemplo, Skype)	UDP ou TCP

2.1.5 Protocolos de camada de aplicação

Acabamos de aprender que processos de rede comunicam-se entre si enviando mensagens para dentro de *sockets*. Mas, como essas mensagens são estruturadas? O que significam os vários campos nas mensagens? Quando os processos enviam as mensagens? Essas perguntas nos transportam para o mundo dos protocolos de camada de aplicação. Um **protocolo de camada de aplicação** define como processos de uma aplicação, que funcionam em sistemas finais diferentes, passam mensagens entre si. Em particular, um protocolo de camada de aplicação define:

- Os tipos de mensagens trocadas, por exemplo, de requisição e de resposta.
- A sintaxe dos vários tipos de mensagens, tais como os campos da mensagem e como os campos são delineados.
- A semântica dos campos, isto é, o significado da informação nos campos.
- Regras para determinar quando e como um processo envia mensagens e responde a mensagens.

Alguns protocolos de camada de aplicação estão especificados em RFCs e, portanto, são de domínio público. Por exemplo, o protocolo de camada de aplicação da Web, HTTP (HyperText Transfer Protocol [RFC 2616]), está à disposição como um RFC. Se um programador de navegador seguir as regras do RFC do HTTP, o navegador estará habilitado a extrair páginas de qualquer servidor que também tenha seguido essas mesmas regras. Muitos outros protocolos de camada de aplicação são próprios e, de modo intencional, não estão disponíveis ao público, por exemplo, o Skype.

É importante distinguir aplicações de rede de protocolos de camada de aplicação, os quais são apenas um pedaço de aplicação de rede (embora muito importante, do nosso ponto de vista!). Examinemos alguns exemplos. A Web é uma aplicação cliente-servidor que permite aos usuários obter documentos de servidores por demanda. A aplicação Web consiste em muitos componentes, entre eles um padrão para formato de documentos (isto é, HTML), navegadores Web (por exemplo, Firefox e Microsoft Internet Explorer), servidores Web (por exemplo, Apache e Microsoft) e um protocolo de camada de aplicação. O HTTP, protocolo de camada de aplicação da Web, define o formato e a sequência das mensagens que são passadas entre o navegador e o servidor. Assim, ele é apenas um pedaço (embora importante) da aplicação Web. Como outro exemplo, a aplicação de correio eletrônico da Internet que também tem muitos componentes, entre eles servidores de correio que armazenam caixas postais de usuários, leitores de correio (como o Microsoft Outlook) que permitem aos usuários ler e criar mensagens, um padrão que define como elas são passadas entre servidores e entre servidores e leitores de correio, e como deve ser interpretado o conteúdo de cabeçalhos de mensagem. O principal protocolo de camada de aplicação para o correio eletrônico é o SMTP (Simple Mail Transfer Protocol) [RFC 5321]. Assim, o SMTP é apenas um pedaço (embora importante) da aplicação de correio eletrônico.

2.1.6 Aplicações de rede abordadas neste livro

Novas aplicações de Internet de domínio público e proprietárias são desenvolvidas todos os dias. Em vez de tratarmos de um grande número dessas aplicações de maneira enciclopédica, preferimos focalizar um pequeno número ao mesmo tempo importantes e populares. Neste capítulo, discutiremos cinco aplicações populares: a Web, a transferência de arquivos, o correio eletrônico, o serviço de diretório e aplicações P2P. Discutiremos primeiro a Web não apenas porque ela é uma aplicação de imensa popularidade, mas também porque seu protocolo de camada de aplicação, HTTP, é direto e fácil de entender. Após estudarmos a Web, examinaremos brevemente o FTP porque ele oferece um ótimo contraste com o HTTP. Em seguida, discutiremos o correio eletrônico, a primeira aplicação de enorme sucesso da Internet. O correio eletrônico é mais complexo do que a Web, pois usa não somente um, mas vários protocolos de camada de aplicação. Após o e-mail, estudaremos o DNS, que provê um serviço de diretório para a Internet. A maioria dos usuários não interage direto com o DNS; em vez disso, eles o chamam indiretamente por meio de outras aplicações (inclusive a Web, a transferência de arquivos e o correio eletrônico). O DNS ilustra de maneira primorosa

como um componente de funcionalidade do núcleo da rede (tradução de nome de rede para endereço de rede) pode ser implementado na camada de aplicação da Internet. Por fim, discutiremos várias aplicações P2P, incluindo aplicações de compartilhamento de arquivos e serviços de busca distribuída. No Capítulo 7, veremos as aplicações de multimídia, incluindo vídeo de fluxo contínuo e voz sobre IP (VoIP).

2.2 A WEB E O HTTP

Até a década de 1990, a Internet era usada principalmente por pesquisadores, acadêmicos e estudantes universitários para efetuar login em hospedeiros remotos, transferir arquivos de hospedeiros locais para remotos e vice-versa, enviar e receber notícias e correio eletrônico. Embora essas aplicações fossem (e continuem a ser) de extrema utilidade, a Internet era desconhecida fora das comunidades acadêmicas e de pesquisa. Então, no início da década de 1990, entrou em cena uma nova aplicação importantíssima — a World Wide Web [Berners-Lee, 1994]. A Web é a aplicação que chamou a atenção do público em geral. Ela transformou drasticamente a maneira como pessoas interagem dentro e fora de seus ambientes de trabalho. Alçou a Internet de apenas mais uma entre muitas para, na essência, a única rede de dados.

Talvez o que mais atraia a maioria dos usuários da Web é que ela funciona *por demanda*. Usuários recebem o que querem, quando querem, o que é diferente da transmissão de rádio e de televisão, que obriga a sintonizar quando o provedor disponibiliza o conteúdo. Além de funcionar por demanda, a Web tem muitas outras características maravilhosas que as pessoas adoram. É muito fácil para qualquer indivíduo fazer que informações fiquem disponíveis na Web — todo mundo pode se transformar em editor a um custo baixíssimo. Hiperenlaces e buscadores nos ajudam a navegar pelo oceano dos sites. Dispositivos gráficos estimulam nossos sentidos. Formulários, applets Java e muitos outros recursos nos habilitam a interagir com páginas e sites. E a Web serve como uma plataforma para muitas aplicações de sucesso que surgiram após 2003, incluindo YouTube, Gmail e Facebook.

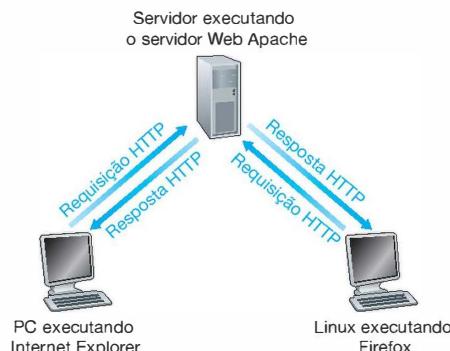
2.2.1 Descrição geral do HTTP

O **HTTP — Protocolo de Transferência de Hipertexto (HyperText Transfer Protocol)** —, o protocolo da camada de aplicação da Web, está no coração da Web e é definido no [RFC 1945] e no [RFC 2616]. O HTTP é executado em dois programas: um cliente e outro servidor. Os dois, executados em sistemas finais diferentes, conversam entre si por meio da troca de mensagens HTTP. O HTTP define a estrutura dessas mensagens e o modo como o cliente e o servidor as trocam. Antes de explicarmos em detalhes o HTTP, devemos revisar a terminologia da Web.

Uma **página Web** (também denominada documento) é constituída de objetos. Um **objeto** é apenas um arquivo — tal como um arquivo HTML, uma imagem JPEG, um applet Java, ou um clipe de vídeo — que se pode acessar com um único URL. A maioria das páginas Web é constituída de um **arquivo-base HTML** e diversos objetos referenciados. Por exemplo, se uma página contiver um texto HTML e cinco imagens JPEG, então ela terá seis objetos: o arquivo-base HTML e mais as cinco imagens. O arquivo-base HTML referencia os outros objetos na página com os URLs dos objetos. Cada URL tem dois componentes: o nome de hospedeiro (*hostname*) do servidor que abriga o objeto e o nome do caminho do objeto. Por exemplo, no URL

```
http://www.someSchool.edu/someDepartment/picture.gif
```

`www.someSchool.edu` é o nome de hospedeiro e `/someDepartment/picture.gif` é o nome do caminho. Como **navegadores Web** (por exemplo, Internet Explorer e Firefox) também executam o lado cliente do HTTP, usaremos as palavras *navegador* e *cliente* indiferentemente nesse contexto. Os **servidores Web**, que executam o lado servidor do HTTP, abrigam objetos Web, cada um endereçado por um URL. São servidores Web populares o Apache e o Microsoft Internet Information Server.

O HTTP define como os clientes requisitam páginas aos servidores e como eles as transferem aos clientes. Discutiremos em detalhes a interação entre cliente e servidor mais adiante, mas a ideia geral está ilustrada na Figura 2.6. Quando um usuário requisita uma página Web (por exemplo, clica sobre um hiperenlace), o navegador envia ao servidor mensagens de requisição HTTP para os objetos da página. O servidor recebe as requisições e responde com mensagens de resposta HTTP que contêm os objetos.

O HTTP usa o TCP como seu protocolo de transporte subjacente (em vez de rodar em cima do UDP). O cliente HTTP primeiro inicia uma conexão TCP com o servidor. Uma vez estabelecida, os processos do navegador e do servidor acessam o TCP por meio de suas interfaces de *socket*. Como descrito na Seção 2.1, no lado cliente a interface *socket* é a porta entre o processo cliente e a conexão TCP; no lado servidor, ela é a porta entre o processo servidor e a conexão TCP. O cliente envia mensagens de requisição HTTP para sua interface *socket* e recebe mensagens de resposta HTTP de sua interface *socket*. De maneira semelhante, o servidor HTTP recebe mensagens de requisição de sua interface *socket* e envia mensagens de resposta para sua interface *socket*. Assim que o cliente envia uma mensagem para sua interface *socket*, a mensagem sai de suas mãos e passa a estar "nas mãos" do TCP. Lembre-se de que na Seção 2.1 dissemos que o TCP oferece ao HTTP um serviço confiável de transferência de dados, o que implica que toda mensagem de requisição HTTP emitida por um processo cliente chegará intacta ao servidor. De maneira semelhante, toda mensagem de resposta HTTP emitida pelo processo servidor chegará intacta ao cliente. Percebemos, nesse ponto, uma das grandes vantagens de uma arquitetura de camadas — o HTTP não precisa se preocupar com dados perdidos ou com detalhes de como o TCP se recupera da perda de dados ou os reordena dentro da rede. Essa é a tarefa do TCP e dos protocolos das camadas mais inferiores da pilha de protocolos.

É importante notar que o servidor envia ao cliente os arquivos solicitados sem armazenar qualquer informação de estado sobre o cliente. Se determinado cliente solicita o mesmo objeto duas vezes em um período de poucos segundos, o servidor não responde dizendo que acabou de enviá-lo; em vez disso, manda de novo o objeto, pois já esqueceu por completo o que fez antes. Como o servidor HTTP não mantém informação alguma sobre clientes, o HTTP é denominado um **protocolo sem estado**. Salientamos também que a Web usa a arquitetura de aplicação cliente-servidor, como descrito na Seção 2.1. Um servidor Web está sempre em funcionamento, tem um endereço IP fixo e atende requisições de potencialmente milhões de navegadores diferentes.

2.2.2 Conexões persistentes e não persistentes

Em muitas aplicações da Internet, o cliente e o servidor se comunicam por um período prolongado de tempo, em que o cliente faz uma série de requisições e o servidor responde a cada uma. Dependendo da aplicação e de como ela está sendo usada, a série de requisições pode ser feita de forma consecutiva, periodicamente

FIGURA 2.6 COMPORTAMENTO DE REQUISIÇÃO-RESPOSTA DO HTTP

em intervalos regulares ou de modo esporádico. Quando a interação cliente-servidor acontece por meio de conexão TCP, o programador da aplicação precisa tomar uma importante decisão — cada par de requisição/resposta deve ser enviado por uma conexão TCP *distinta* ou todas as requisições e suas respostas devem ser enviadas por uma *mesma* conexão TCP? Na abordagem anterior, a aplicação utiliza **conexões não persistentes**; e na última abordagem, **conexões persistentes**. Para entender melhor este assunto, vamos analisar as vantagens e desvantagens das conexões não persistentes e das conexões persistentes no contexto de uma aplicação específica, o HTTP, que pode utilizar as duas. Embora o HTTP utilize conexões persistentes em seu modo padrão, os clientes e servidores HTTP podem ser configurados para utilizar a não persistente.

O HTTP com conexões não persistentes

Vamos percorrer as etapas da transferência de uma página de um servidor para um cliente para o caso de conexões não persistentes. Suponhamos que uma página consista em um arquivo-base HTML e em dez imagens JPEG e que todos esses 11 objetos residam no mesmo servidor. Suponha também que o URL para o arquivo-base HTTP seja

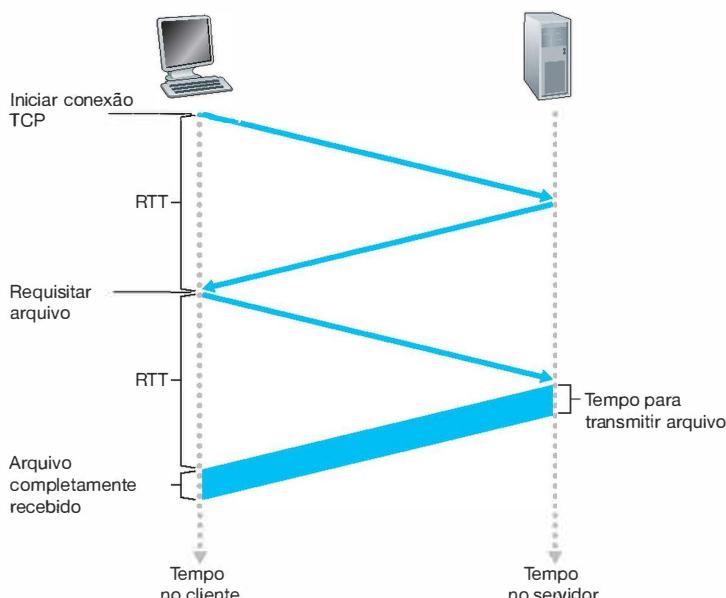
```
http://www.someSchool.edu/someDepartment/home.index
```

Eis o que acontece:

1. O processo cliente HTTP inicia uma conexão TCP para o servidor `www.someSchool.edu` na porta número 80, que é o número de porta *default* para o HTTP. Associados à conexão TCP, haverá um *socket* no cliente e um *socket* no servidor.
2. O cliente HTTP envia uma mensagem de requisição HTTP ao servidor por meio de seu *socket*. Essa mensagem inclui o nome de caminho `/someDepartment/home.index`. (Discutiremos mensagens HTTP mais detalhadamente logo adiante.)
3. O processo servidor HTTP recebe a mensagem de requisição por meio de seu *socket*, extrai o objeto `/someDepartment/home.index` de seu armazenamento (RAM ou disco), encapsula-o em uma mensagem de resposta HTTP e a envia ao cliente pelo *socket*.
4. O processo servidor HTTP ordena ao TCP que encerre a conexão TCP. (Mas, na realidade, o TCP só a encerrará quando tiver certeza de que o cliente recebeu a mensagem de resposta intacta.)
5. O cliente HTTP recebe a mensagem de resposta e a conexão TCP é encerrada. A mensagem indica que o objeto encapsulado é um arquivo HTML. O cliente extrai o arquivo da mensagem de resposta, analisa o arquivo HTML e encontra referências aos dez objetos JPEG.
6. As primeiras quatro etapas são repetidas para cada um dos objetos JPEG referenciados.

À medida que recebe a página Web, o navegador a apresenta ao usuário. Dois navegadores diferentes podem interpretar (isto é, exibir ao usuário) uma página de modos um pouco diferentes. O HTTP não tem nada a ver com o modo como uma página Web é interpretada por um cliente. As especificações do HTTP ([RFC 1945] e [RFC 2616]) definem apenas o protocolo de comunicação entre o programa cliente HTTP e o programa servidor HTTP.

As etapas apresentadas ilustram a utilização de conexões não persistentes, nas quais cada conexão TCP é encerrada após o servidor enviar o objeto — a conexão não persiste para outros objetos. Note que cada conexão TCP transporta exatamente uma mensagem de requisição e uma mensagem de resposta. Assim, nesse exemplo, quando um usuário solicita a página Web, são geradas 11 conexões TCP.

Nos passos descritos, fomos intencionalmente vagos sobre se os clientes obtêm as dez JPEGs por meio de dez conexões TCP em série ou se algumas delas são recebidas por conexões TCP paralelas. Na verdade, usuários podem configurar navegadores modernos para controlar o grau de paralelismo. Nos modos *default*, a maioria dos navegadores abre de cinco a dez conexões TCP paralelas e cada uma manipula uma transação requisição/resposta. Se o usuário preferir, o número máximo de conexões paralelas poderá ser fixado em um, caso em que as

dez conexões são estabelecidas em série. Como veremos no próximo capítulo, a utilização de conexões paralelas reduz o tempo de resposta.

Antes de continuar, vamos fazer um rascunho de cálculo para estimar o tempo que transcorre entre a requisição e o recebimento de um arquivo-base HTTP por um cliente. Para essa finalidade, definimos o **tempo de viagem de ida e volta** (*round-trip time* — RTT), ou seja, o tempo que leva para um pequeno pacote viajar do cliente ao servidor e de volta ao cliente. O RTT inclui atrasos de propagação de pacotes, de fila de pacotes em roteadores e comutadores intermediários e de processamento de pacotes. (Esses atrasos foram discutidos na Seção 1.4.) Considere, agora, o que acontece quando um usuário clica sobre um hiperenlace. Como ilustrado na Figura 2.7, isso faz com que o navegador inicie uma conexão TCP entre ele e o servidor, o que envolve uma "apresentação de três vias" — o cliente envia um pequeno segmento TCP ao servidor, este o reconhece e responde com um pequeno segmento ao cliente que, por fim, o reconhece novamente para o servidor. As duas primeiras partes da apresentação de três vias representam um RTT. Após concluí-las, o cliente envia a mensagem de requisição HTTP combinada com a terceira parte da apresentação de três vias (o reconhecimento) por meio da conexão TCP. Assim que a mensagem de requisição chega ao servidor, este envia o arquivo HTML por meio da conexão TCP. Essa requisição/resposta HTTP consome outro RTT. Assim, de modo aproximado, o tempo total de resposta são dois RTTs mais o tempo de transmissão do arquivo HTML no servidor.

O HTTP com conexões persistentes

Conexões não persistentes têm algumas desvantagens. Primeiro, uma nova conexão deve ser estabelecida e mantida para *cada objeto solicitado*. Para cada conexão, devem ser alocados buffers TCP e conservadas variáveis TCP tanto no cliente quanto no servidor. Isso pode sobrecarregar seriamente o servidor Web, que poderá estar processando requisições de centenas de diferentes clientes ao mesmo tempo. Segundo, como acabamos de descrever, cada objeto sofre dois RTTs — um RTT para estabelecer a conexão TCP e outro para solicitar e receber um objeto.

FIGURA 2.7 CÁLCULO SIMPLES PARA O TEMPO NECESSÁRIO PARA SOLICITAR E RECEBER UM ARQUIVO HTML

Em conexões persistentes, o servidor deixa a conexão TCP aberta após enviar resposta. Requisições e respostas subsequentes entre os mesmos cliente e servidor podem ser enviadas por meio da mesma conexão. Em particular, uma página Web inteira (no exemplo anterior, o arquivo-base HTML e as dez imagens) pode ser enviada mediante uma única conexão TCP persistente. Além do mais, várias páginas residindo no mesmo servidor podem ser enviadas ao mesmo cliente por uma única conexão TCP persistentes. Essas requisições por objetos podem ser feitas em sequência sem ter de esperar por respostas a requisições pendentes (paralelismo). Em geral, o servidor HTTP fecha uma conexão quando ela não é usada durante certo tempo (um intervalo de pausa configurável). Quando o servidor recebe requisições consecutivas, os objetos são enviados de forma ininterrupta. O modo *default* do HTTP usa conexões persistentes com paralelismo. Faremos uma comparação quantitativa entre os desempenhos de conexões persistentes e não persistentes nos exercícios de fixação dos Capítulos 2 e 3. Aconselhamos o leitor interessado a consultar Heidemann [1997] e Nielsen [1997].

2.2.3 Formato da mensagem HTTP

As especificações do HTTP [RFC 1945; RFC 2616] incluem as definições dos formatos das mensagens HTTP. Há dois tipos delas: de requisição e de resposta, ambas discutidas a seguir.

Mensagem de requisição HTTP

Apresentamos a seguir uma mensagem de requisição HTTP típica:

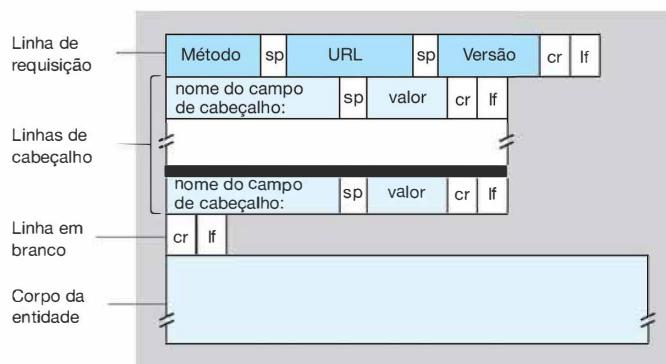
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

Podemos aprender bastante examinando essa simples mensagem de requisição. Primeiro, vemos que ela está escrita em texto ASCII comum, de modo que pode ser lida por qualquer um que conheça computadores. Segundo, vemos que ela é constituída de cinco linhas, cada qual seguida de um '*carriage return*' e '*line feed*' (fim de linha). A última linha é seguida de um comando adicional de '*carriage return*' e '*line feed*'. Embora essa tenha cinco linhas, uma mensagem de requisição pode ter muito mais e também menos do que isso, podendo conter até mesmo uma única linha. A primeira linha é denominada **linha de requisição**; as subsequentes são denominadas **linhas de cabeçalho**. A linha de requisição tem três campos: o do método, o do URL e o da versão do HTTP. O campo do método pode assumir vários valores diferentes, entre eles `GET`, `POST`, `HEAD`, `PUT` e `DELETE`. A grande maioria das mensagens de requisição HTTP emprega o método GET, o qual é usado quando o navegador requisita um objeto e este é identificado no campo do URL. Nesse exemplo, o navegador está requisitando o objeto `/somedir/page.html`. A versão é autoexplicativa. Nesse exemplo, o navegador executa a versão HTTP/1.1.

Vamos agora examinar as linhas de cabeçalho do exemplo. A linha de cabeçalho Host:www.some-school.edu especifica o hospedeiro no qual o objeto reside. Talvez você ache que ela é desnecessária, pois já existe uma conexão TCP para o hospedeiro. Mas, como veremos na Seção 2.2.5, a informação fornecida pela linha de cabeçalho do hospedeiro é exigida por *caches proxy* da Web. Ao incluir a linha de cabeçalho `Connection: close`, o navegador está dizendo ao servidor que não quer usar conexões persistentes; quer que o servidor feche a conexão após o envio do objeto requisitado. A linha de cabeçalho `User-agent`: especifica o agente de usuário, isto é, o tipo de navegador que está fazendo a requisição ao servidor. Neste caso, o agente de usuário é o Mozilla/5.0, um navegador Firefox. Essa linha de cabeçalho é útil porque, na verdade, o servidor pode enviar versões diferentes do mesmo objeto a tipos diferentes de agentes de usuário. (Cada versão é endereçada pelo mesmo URL.) Por fim, o cabeçalho `Accept-language`: mostra que o usuário prefere receber uma versão em francês do objeto se este existir no servidor; se não existir, o servidor deve enviar a versão *default*. O cabeçalho `Accept-language`: é apenas um dos muitos de negociação de conteúdo disponíveis no HTTP.

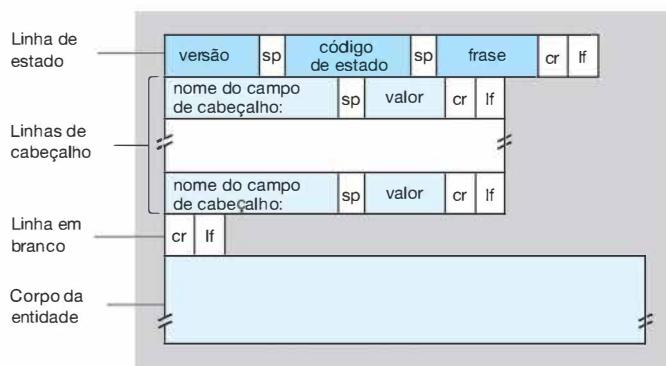
Após examinar um exemplo, vamos agora analisar o formato geral de uma mensagem de requisição, ilustrado na Figura 2.8. Vemos que tal formato é muito parecido com nosso exemplo anterior. Contudo, você provavelmente notou que, após as linhas de cabeçalho (e após a linha adicional com "*carriage return*" e "*line feed*"), há um "corpo de entidade". O corpo de entidade fica vazio com o método `GET`, mas é utilizado com o método `POST`. Um cliente HTTP em geral usa o método `POST` quando o usuário preenche um formulário — por exemplo, quando fornece palavras de busca a um site buscador. Com uma mensagem `POST`, o usuário continua solicitando uma página Web ao servidor, mas seu conteúdo depende do que ele escreveu nos campos do formulário. Se o valor do campo de método for `POST`, então o corpo de entidade conterá o que o usuário digitou nos campos do formulário.

Seríamos omissos se não mencionássemos que uma requisição gerada com um formulário não utiliza necessariamente o método `POST`. Ao contrário, formulários HTML costumam empregar o método `GET` e incluem os dados digitados (nos campos do formulário) no URL requisitado. Por exemplo, se um formulário usar o método `GET`, tiver dois campos e as entradas desses dois campos forem `monkeys` e `bananas`, então a estrutura do URL será `www.somesite.com/animalsearch?monkeys&bananas`. Ao navegar normalmente pela Web, você talvez já tenha notado URLs extensos como esse.

O método `HEAD` é semelhante ao `GET`. Quando um servidor recebe uma requisição com o método `HEAD`, responde com uma mensagem HTTP, mas deixa de fora o objeto requisitado. Esse método é usado com frequência pelos programadores de aplicação para depuração. O método `PUT` é muito usado junto com ferramentas de edição da Web. Permite que um usuário carregue um objeto para um caminho (diretório) específico em um servidor Web específico. O método `PUT` também é usado por aplicações que precisam carregar objetos para servidores Web. O método `DELETE` permite que um usuário, ou uma aplicação, elimine um objeto em um servidor Web.

Mensagem de resposta HTTP

Apresentamos a seguir uma mensagem de resposta HTTP típica. Essa mensagem poderia ser a resposta ao exemplo de mensagem de requisição que acabamos de discutir.

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 09 Aug 2011 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(dados dados dados dados dados ...)
```

FIGURA 2.8 FORMATO GERAL DE UMA MENSAGEM DE REQUISIÇÃO HTTP

Vamos examinar com cuidado essa mensagem de resposta. Ela tem três seções: uma linha inicial ou **linha de estado**, seis **linhas de cabeçalho** e, em seguida, o **corpo da entidade**, que é a parte principal da mensagem — contém o objeto solicitado (representado por `dados dados dados dados dados` ...). A linha de estado tem três campos: o de versão do protocolo, um código de estado e uma mensagem de estado correspondente. Neste exemplo, ela mostra que o servidor está usando o HTTP/1.1 e que está tudo OK (isto é, o servidor encontrou e está enviando o objeto solicitado).

Agora, vamos ver as linhas de cabeçalho. O servidor usa `Connection: close` para informar ao cliente que fechará a conexão TCP após enviar a mensagem. A linha de cabeçalho `Date`: indica a hora e a data em que a resposta HTTP foi criada e enviada pelo servidor. Note que esse não é o horário em que o objeto foi criado nem o de sua modificação mais recente; é a hora em que o servidor extraiu o objeto de seu sistema de arquivos, inseriu-o na mensagem de resposta e a enviou. A linha de cabeçalho `Server:` mostra que a mensagem foi gerada por um servidor Web Apache, semelhante à linha de cabeçalho `User-agent:` na mensagem de requisição HTTP. A linha de cabeçalho `Last-Modified`: indica a hora e a data em que o objeto foi criado ou sofreu a última modificação. Esse cabeçalho, que logo estudaremos em mais detalhes, é fundamental para fazer *cache* do objeto, tanto no cliente local quanto em servidores de *cache* da rede (também conhecidos como servidores *proxy*). A linha de cabeçalho `Content-Length`: indica o número de bytes do objeto que está sendo enviado e a linha `Content-Type`: mostra que o objeto presente no corpo da mensagem é um texto HTML. (O tipo do objeto é oficialmente indicado pelo cabeçalho `Content-Type`: e não pela extensão do arquivo.)

Após examinar um exemplo, vamos agora analisar o formato geral de uma mensagem de resposta, ilustrado na Figura 2.9. Esse formato geral de mensagem de resposta condiz com o exemplo anterior de uma mensagem de resposta. Mas falemos um pouco mais sobre códigos de estado e suas frases, que indicam o resultado da requisição. Eis alguns códigos de estado e frases associadas comuns:

- `200 OK`: requisição bem-sucedida e a informação é entregue com a resposta.
- `301 Moved Permanently`: objeto requisitado foi removido em definitivo; novo URL é especificado no cabeçalho Location: da mensagem de resposta. O software do cliente recuperará automaticamente o novo URL.
- `400 Bad Request`: código genérico de erro que indica que a requisição não pôde ser entendida pelo servidor.
- `404 Not Found`: o documento requisitado não existe no servidor.
- `505 HTTP Version Not Supported`: a versão do protocolo HTTP requisitada não é suportada pelo servidor.

FIGURA 2.9 FORMATO GERAL DE UMA MENSAGEM DE RESPOSTA HTTP

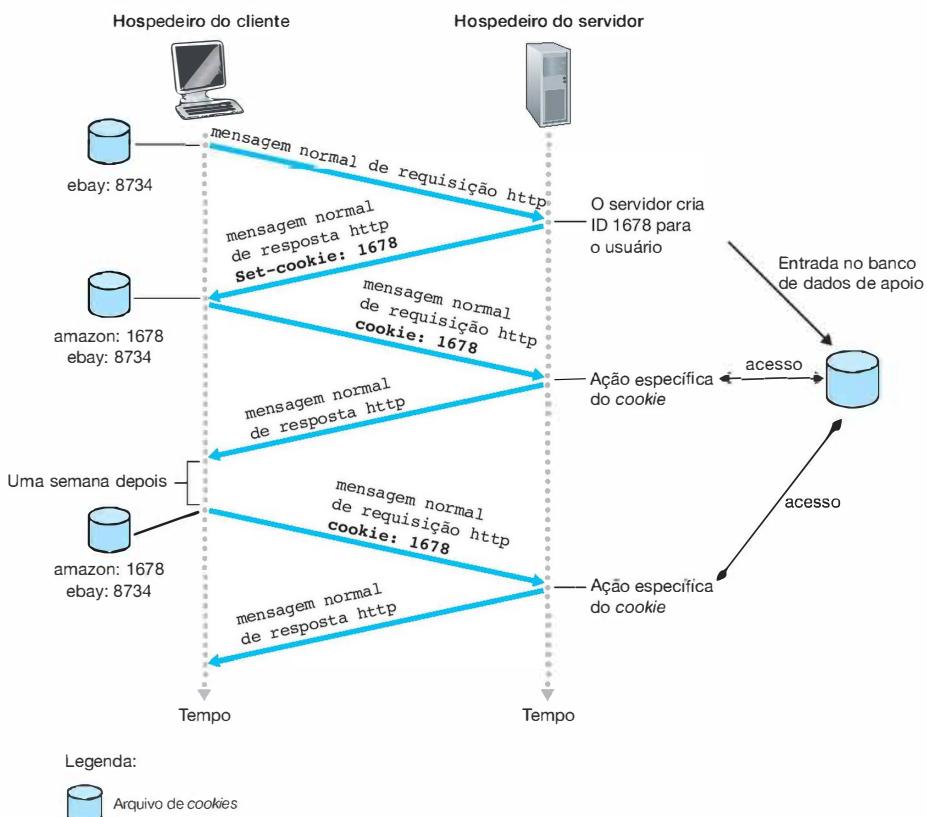
Você gostaria de ver uma mensagem de resposta HTTP real? É muito recomendável e muito fácil! Primeiro, dê um comando Telnet em seu servidor favorito. Em seguida, digite uma mensagem de requisição de uma linha solicitando algum objeto abrigado no servidor. Por exemplo, se você tem acesso a um prompt de comando, digite:

```
telnet cis.poly.edu 80
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

(Pressione duas vezes a tecla "Enter" após digitar a última linha.) Essa sequência de comandos abre uma conexão TCP para a porta número 80 do hospedeiro `cis.poly.edu` e, em seguida, envia a mensagem de requisição HTTP. Deverá aparecer uma mensagem de resposta que inclui o arquivo-base HTML da homepage do professor Ross. Se você preferir apenas ver as linhas da mensagem HTTP e não receber o objeto em si, substitua `GET` por `HEAD`. Finalmente, substitua `/~ross/ por /~banana/` e veja que tipo de mensagem você obtém.

Nesta seção, discutimos várias linhas de cabeçalho que podem ser usadas em mensagens de requisição e de resposta HTTP. A especificação do HTTP define muitas outras linhas de cabeçalho que podem ser inseridas por navegadores, servidores Web e servidores de *cache* da rede. Vimos apenas um pouco do total de linhas de cabeçalho. Examinaremos mais algumas a seguir e mais um pouco quando discutirmos armazenagem Web na Seção 2.2.5. Uma discussão muito abrangente e fácil de ler sobre o protocolo HTTP, seus cabeçalhos e códigos de estado, pode ser encontrada em Krishnamurty [2001].

Como um navegador decide quais linhas de cabeçalho serão incluídas em uma mensagem de requisição? Como um servidor Web decide quais linhas de cabeçalho serão incluídas em uma mensagem de resposta? Um navegador vai gerar linhas de cabeçalho em função de seu tipo e versão (por exemplo, um navegador HTTP/1.0 não vai gerar nenhuma linha de cabeçalho 1.1), da configuração do usuário para o navegador (por exemplo, idioma preferido) e se o navegador tem uma versão do objeto possivelmente desatualizada em sua memória. Servidores Web se comportam de maneira semelhante: há diferentes produtos, versões e configurações, e todos influenciam as linhas de cabeçalho que são incluídas nas mensagens de resposta.

2.2.4 Interação usuário-servidor: *cookies*

Mencionamos anteriormente que um servidor HTTP não tem estado, o que simplifica o projeto do servidor e vem permitindo que engenheiros desenvolvam servidores Web de alto desempenho que podem manipular milhares de conexões TCP simultâneas. No entanto, é sempre bom que um site identifique usuários, seja porque o servidor deseja restringir acesso, seja porque quer apresentar conteúdo em função da identidade do usuário. Para essas finalidades, o HTTP usa *cookies*. *Cookies*, definidos no [RFC 6265], permitem que sites monitorem seus usuários. Hoje, a maioria dos sites comerciais utiliza *cookies*.

Como ilustrado na Figura 2.10, a tecnologia dos *cookies* tem quatro componentes: (1) uma linha de cabeçalho de *cookie* na mensagem de resposta HTTP; (2) uma linha de cabeçalho de *cookie* na mensagem de requisição HTTP; (3) um arquivo de *cookie* mantido no sistema final do usuário e gerenciado pelo navegador do usuário; (4) um banco de dados de apoio no site. Utilizando a Figura 2.10, vamos esmiuçar um exemplo de como os *cookies* são usados. Suponha que Susan, que sempre acessa a Web usando o Internet Explorer de seu PC, acesse o Amazon.com pela primeira vez, e que, no passado, ela já tenha visitado o site da eBay. Quando a requisição chega ao servidor da Amazon, ele cria um número de identificação exclusivo e uma entrada no seu banco de dados de apoio, que é indexado pelo número de identificação. Então, o servidor da Amazon responde ao navegador de Susan, incluindo na resposta HTTP um cabeçalho `Set-cookie`: que contém o número de identificação. Por exemplo, a linha de cabeçalho poderia ser:

```
Set-cookie: 1678
```

Quando recebe a mensagem de resposta HTTP, o navegador de Susan vê o cabeçalho `Set-cookie`: e, então, anexa uma linha ao arquivo especial de *cookies* que ele gerencia. Essa linha inclui o nome de hospedeiro do

FIGURA 2.10 MANTENDO O ESTADO DO USUÁRIO COM *COOKIES*

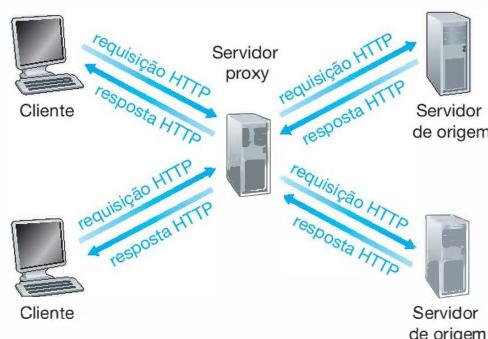
servidor e seu número de identificação no cabeçalho. Observe que o arquivo de *cookie* já possui um entrada para o eBay, pois Susan já visitou esse site no passado. Toda vez que ela requisita uma página enquanto navega pelo site da Amazon, seu navegador consulta seu arquivo de *cookies*, extrai seu número de identificação para esse site e insere na requisição HTTP uma linha de cabeçalho de *cookie* que inclui o número de identificação. Especificamente, cada uma de suas requisições HTTP ao servidor da Amazon inclui a linha de cabeçalho:

```
Cookie: 1678
```

Dessa maneira, o servidor da Amazon pode monitorar a atividade de Susan em seu site e, embora não saiba necessariamente que o nome dela é Susan, sabe com exatidão quais páginas o usuário 1678 visitou, em que ordem e em que horários! Então, pode utilizar *cookies* para oferecer um serviço de carrinho de compra — a Amazon pode manter uma lista de todas as compras de Susan, de modo que ela possa pagar por todas elas ao mesmo tempo, no final da sessão.

Se Susan voltar ao site da Amazon, digamos, uma semana depois, seu navegador continuará a inserir a linha de cabeçalho `Cookie: 1678` nas mensagens de requisição. A Amazon pode recomendar produtos com base nas páginas que Susan visitou anteriormente. Se ela também se registrar no site — fornecendo seu nome completo, endereço de e-mail, endereço postal e informações de cartão de crédito — a Amazon pode incluir essas informações no banco de dados e, assim, associar o nome de Susan com seu número de identificação (e

com todas as páginas que ela consultou em suas visitas anteriores). É assim que a Amazon e outros sites de comércio eletrônico oferecem "compras com um só clique" — quando quiser comprar um item em uma visita posterior, Susan não precisará digitar de novo seu nome, número de cartão de crédito, nem endereço.

Essa discussão nos mostrou que *cookies* podem ser usados para identificar um usuário. Quando visitar um site pela primeira vez, um usuário pode fornecer dados de identificação (possivelmente seu nome). No decorrer das próximas sessões, o navegador passa um cabeçalho de *cookie* ao servidor durante todas as visitas subsequentes ao site, identificando, desse modo, o usuário ao servidor. Assim, vemos que os *cookies* podem ser usados para criar uma camada de sessão de usuário sobre HTTP sem estado. Por exemplo, quando um usuário acessa uma aplicação de e-mail baseada na Web (como o Hotmail), o navegador envia informações de *cookie* ao servidor, permitindo que o servidor identifique o usuário por meio da sessão deste com a aplicação.

Embora *cookies* quase sempre simplifiquem a experiência de compra pela Internet, continuam provocando muita controvérsia porque também podem ser considerados invasão da privacidade do usuário. Como acabamos de ver, usando uma combinação de *cookies* e informações de conta fornecidas pelo usuário, um site pode ficar sabendo muita coisa sobre esse usuário e, potencialmente, vender o que sabe para algum terceiro. O Cookie Central [Cookie Central, 2012] inclui muitas informações sobre a controvérsia dos *cookies*.

2.2.5 *Caches* Web

Um ***cache* Web** — também denominado **servidor *proxy*** — é uma entidade da rede que atende requisições HTTP em nome de um servidor Web de origem. O *cache* Web tem seu próprio disco de armazenagem e mantém, dentro dele, cópias de objetos recentemente requisitados. Como ilustrado na Figura 2.11, o navegador de um usuário pode ser configurado de modo que todas as suas requisições HTTP sejam dirigidas primeiro ao *cache* Web. Uma vez que esteja configurado um navegador, cada uma das requisições de um objeto que o navegador faz é primeiro dirigida ao *cache* Web. Como exemplo, suponha que um navegador esteja requisitando o objeto `http://www.someschool.edu/campus.gif`. Eis o que acontece:

1. O navegador estabelece uma conexão TCP com o *cache* Web e envia a ele uma requisição HTTP para o objeto.
2. O *cache* Web verifica se tem uma cópia do objeto armazenada localmente. Se tiver, envia o objeto ao navegador do cliente, dentro de uma mensagem de resposta HTTP.
3. Se não tiver o objeto, o *cache* Web abre uma conexão TCP com o servidor de origem, isto é, com `www.someschool.edu`. Então, envia uma requisição HTTP do objeto para a conexão TCP. Após recebê-la, o servidor de origem envia o objeto ao *cache* Web, dentro de uma resposta HTTP.

FIGURA 2.11 CLIENTES REQUISITANDO OBJETOS POR MEIO DE UM *CACHE* WEB

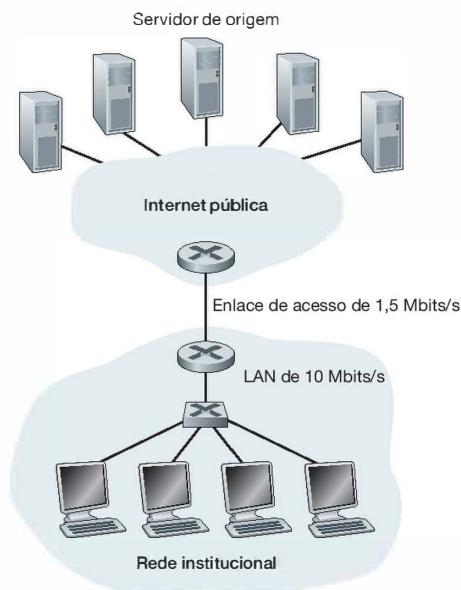
4. Quando recebe o objeto, o *cache* Web guarda uma cópia em seu armazenamento local e envia outra, dentro de uma mensagem de resposta HTTP, ao navegador do cliente (pela conexão TCP existente entre o navegador do cliente e o *cache* Web).

Note que um *cache* é, ao mesmo tempo, um servidor e um cliente. Quando recebe requisições de um navegador e lhe envia respostas, é um servidor. Quando envia requisições para um servidor de origem e recebe respostas dele, é um cliente.

Em geral, é um ISP que compra e instala um *cache* Web. Por exemplo, uma universidade poderia instalar um *cache* na rede de seu *campus* e configurar todos os navegadores apontando para esse *cache*. Ou um importante ISP residencial (como a AOL) poderia instalar um ou mais *caches* em sua rede e configurar antecipadamente os navegadores que fornece apontando para os *caches* instalados.

O *cache* na Web tem tido ampla utilização na Internet por duas razões. Primeiro, pode reduzir substancialmente o tempo de resposta para a requisição de um cliente, em particular se o gargalo da largura de banda entre o cliente e o servidor de origem for muito menor do que aquele entre o cliente e o *cache*. Se houver uma conexão de alta velocidade entre o cliente e o *cache*, como em geral é o caso, e se este tiver o objeto requisitado, então ele poderá entregar com rapidez o objeto ao cliente. Segundo, como logo ilustraremos com um exemplo, *caches* Web podem reduzir de modo substancial o tráfego no enlace de acesso de uma instituição qualquer à Internet. Com a redução do tráfego, a instituição (por exemplo, uma empresa ou uma universidade) não precisa ampliar sua largura de banda tão rapidamente, o que diminui os custos. Além disso, *caches* Web podem reduzir bastante o tráfego na Internet como um todo, melhorando, assim, o desempenho para todas as aplicações.

Para entender melhor os benefícios dos *caches*, vamos considerar um exemplo no contexto da Figura 2.12. Essa figura mostra duas redes: uma rede institucional e a Internet pública. A rede institucional é uma LAN de alta velocidade. Um roteador da rede institucional e um roteador da Internet estão ligados por um enlace de 15 Mbits/s. Os servidores de origem estão todos ligados à Internet, mas localizados pelo mundo todo. Suponha que o tamanho médio do objeto seja 1 Mbit/s e que a taxa média de requisição dos navegadores da instituição até os servidores de origem seja de 15 requisições por segundo. Imagine também que o tamanho das mensagens de requisição HTTP seja insignificante e, portanto, elas não criem tráfego nas redes ou no enlace de acesso (do roteador da instituição até o da Internet). Suponha ainda que o tempo entre o envio de uma requisição HTTP (dentro de um datagrama IP) pelo roteador do lado da Internet do enlace de acesso mostrado na Figura 2.12 e o recebimento da resposta (em geral, dentro de muitos datagramas IPs) seja de 2 s em média. Esse último atraso é denominado informalmente "atraso da Internet".

O tempo de resposta total — isto é, aquele transcorrido entre a requisição de um objeto feita pelo navegador e o recebimento dele — é a soma do atraso da LAN, do atraso de acesso (isto é, o atraso entre os dois roteadores) e do atraso da Internet. Vamos fazer agora um cálculo bastante rudimentar para estimar esse atraso. A intensidade de tráfego na LAN (veja a Seção 1.4.2) é

$$(15 \text{ requisições/s}) \cdot (1 \text{ Mbit/s/requisição})/(100 \text{ Mbits/s}) = 0{,}15$$

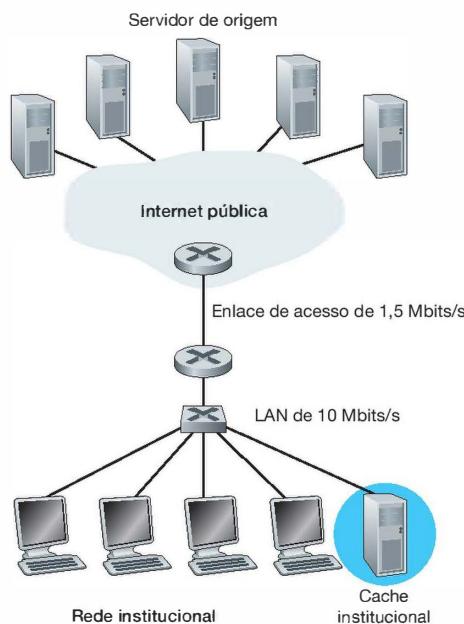
ao passo que a intensidade de tráfego no enlace de acesso (do roteador da Internet ao da instituição) é

$$(15 \text{ requisições/s}) = (1 \text{ Mbit/s/requisição})/(15 \text{ Mbits/s}) = 1$$

Uma intensidade de tráfego de 0,15 em uma LAN resulta em, no máximo, dezenas de milissegundos de atraso; por conseguinte, podemos desprezar o atraso da LAN. Contudo, como discutimos na Seção 1.4.2, à medida que a intensidade de tráfego se aproxima de 1 (como é o caso do enlace de acesso da Figura 2.12), o atraso em um enlace se torna muito grande e cresce sem limites. Assim, o tempo médio de resposta para atender requisições será da ordem de minutos, se não for maior, o que é inaceitável para os usuários da instituição. Claro, algo precisa ser feito.

Uma possível solução seria aumentar a velocidade de acesso de 15 Mbits/s para, digamos, 100 Mbits/s. Isso reduziria a intensidade de tráfego no enlace de acesso a 0,15, o que se traduziria em atrasos desprezíveis entre os dois roteadores. Nesse caso, o tempo total de resposta seria de mais ou menos 2 s, isto é, o atraso da Internet. Mas essa solução também significa que a instituição tem de atualizar seu enlace de acesso de 15 Mbits/s para 100 Mbits/s, o que pode ser muito dispendioso.

FIGURA 2.12 GARGALO ENTRE UMA REDE INSTITUCIONAL E A INTERNET

Considere agora a solução alternativa de não atualizar o enlace de acesso, mas, em vez disso, instalar um *cache* Web na rede institucional. Essa solução é ilustrada na Figura 2.13. A taxa de resposta local — a fração de requisições atendidas por um *cache* — em geral fica na faixa de 0,2 a 0,7 na prática. Para fins de ilustração, vamos supor que a taxa de resposta local do *cache* dessa instituição seja 0,4. Como os clientes e o *cache* estão conectados à mesma LAN de alta velocidade, 40% das requisições serão atendidas quase de imediato pelo *cache*, digamos, em 10 ms. Mesmo assim, os demais 60% das requisições ainda precisam ser atendidos pelos servidores de origem. Mas, com apenas 60% dos objetos requisitados passando pelo enlace de acesso, a intensidade de tráfego neste diminui de 1,0 para 0,6. Em geral, uma intensidade de tráfego menor do que 0,8 corresponde a um atraso pequeno, digamos, dezenas de milissegundos, no caso de um enlace de 15 Mbits/s. Esse atraso é desprezível se comparado aos 2 s do atraso da Internet. Dadas essas considerações, o atraso médio é, portanto,

$$0{,}4 \cdot (0{,}01 \text{ s}) + 0{,}6 \cdot (2{,}01 \text{ s})$$

que é ligeiramente maior do que 1,2 s. Assim, essa segunda solução resulta em tempo de resposta até menor do que o da primeira e não requer que a instituição atualize seu enlace com a Internet. Evidentemente, a instituição terá de comprar e instalar um *cache* Web. Mas esse custo é baixo — muitos *caches* usam softwares de domínio público que rodam em PCs baratos.

Com o uso de **redes de distribuição de conteúdo (CDNs)**, *caches* Web estão cada vez mais desempenhando um papel importante na Internet. Uma empresa de CDN instala muitos *caches* geograficamente dispersos pela Internet, localizando assim grande parte do tráfego. Existem CDNs compartilhadas (como Akamai e Limelight) e CDNs dedicadas (como Google e Microsoft). Discutiremos as CDNs mais em detalhes no Capítulo 7.

2.2.6 GET condicional

Embora possa reduzir os tempos de resposta do ponto de vista do usuário, fazer *cache* introduz um novo problema — a cópia de um objeto existente no *cache* pode estar desatualizada. Em outras palavras, o objeto

FIGURA 2.13 ACRESCENTANDO UM *CACHE* À REDE INSTITUCIONAL

abrigado no servidor pode ter sido modificado desde a data em que a cópia entrou no *cache* no cliente. Felizmente, o HTTP tem um mecanismo que permite que um *cache* verifique se seus objetos estão atualizados. Esse mecanismo é denominado **GET condicional** (*conditional GET*). Uma mensagem de requisição HTTP é denominada uma mensagem GET condicional se (1) usar o método GET e (2) possuir uma linha de cabeçalho `If-Modified-Since:`.

Para ilustrar como o GET condicional funciona, vamos examinar um exemplo. Primeiro, um *cache proxy* envia uma mensagem de requisição a um servidor em nome de um navegador requisitante:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
```

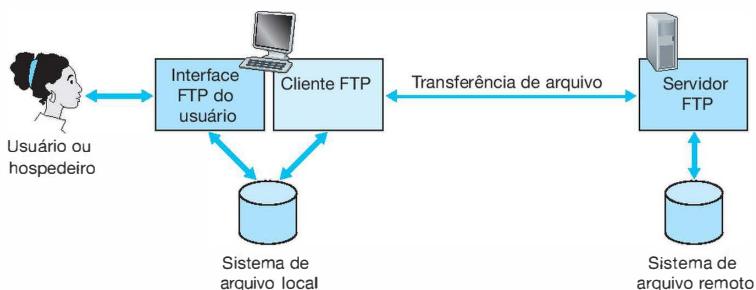
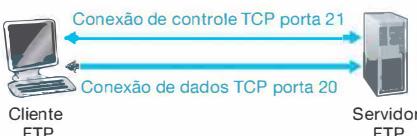
Segundo, o servidor Web envia ao *cache* uma mensagem de resposta com o objeto requisitado:

```
HTTP/1.1 200 OK
Date: Sat, 8 Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 7 Sep 2011 09:23:24
Content-Type: image/gif

(dados dados dados dados dados ...)
```

O *cache* encaminha o objeto ao navegador requisitante, mas também o guarda em sua memória *cache* local. O importante é que ele também guarda, junto com o objeto, a data da última modificação. Terceiro, uma semana depois, outro navegador requisita ao *cache* o mesmo objeto, que ainda está ali. Como esse objeto pode ter sido modificado no servidor na semana anterior, o navegador realiza uma verificação de atualização emitindo um GET condicional. Especificamente, o *cache* envia:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 7 Sep 2011 09:23:24
```

Note que o valor da linha de cabeçalho `If-modified-since:` é idêntico ao da linha de cabeçalho `Last-Modified:` que foi enviada pelo servidor há uma semana. Esse GET condicional está dizendo ao servidor para enviar o objeto somente se ele tiver sido modificado desde a data especificada. Suponha que o objeto não tenha sofrido modificação desde 7 set. 2011 09:23:24. Então, em quarto lugar, o servidor Web envia uma mensagem de resposta ao *cache*:

```
HTTP/1.1 304 Not Modified
Date: Sat, 15 Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)

(corpo de mensagem vazio)
```

Vemos que, em resposta ao GET condicional, o servidor ainda envia uma mensagem de resposta, mas não inclui nela o objeto requisitado, o que apenas desperdiçaria largura de banda e aumentaria o tempo de resposta do ponto de vista do usuário, em particular se o objeto fosse grande. Note que, na linha de estado dessa última mensagem de resposta está inserido 304 `Not Modified`, que informa ao *cache* que ele pode seguir e transmitir ao navegador requisitante a cópia do objeto que está contida nele (no *cache* proxy).

Assim, finalizamos nossa discussão sobre HTTP, o primeiro protocolo da Internet (um protocolo da camada de aplicação) que estudamos em detalhes. Vimos o formato das mensagens HTTP e as ações tomadas pelo cliente e servidor Web quando essas mensagens são enviadas e recebidas. Também estudamos um pouco da infraestrutura da aplicação da Web, incluindo *caches*, *cookies* e banco de dados de apoio, todos associados, de algum modo, ao protocolo HTTP.

2.3 TRANSFERÊNCIA DE ARQUIVO: FTP

Em uma sessão FTP típica, o usuário, sentado à frente de um hospedeiro (o local), quer transferir arquivos de ou para um hospedeiro remoto. Para acessar a conta remota, o usuário deve fornecer uma identificação e uma senha. Após fornecer essas informações de autorização, pode transferir arquivos do sistema local de arquivos para o sistema remoto e vice-versa. Como mostra a Figura 2.14, o usuário interage com o FTP por meio de um agente de usuário FTP. Primeiro, ele fornece o nome do hospedeiro remoto, o que faz com que o processo cliente FTP do hospedeiro local estabeleça uma conexão TCP com o processo servidor FTP do hospedeiro remoto. O usuário então fornece sua identificação e senha, que são enviadas pela conexão TCP como parte dos comandos FTP. Assim que autorizado pelo servidor, o usuário copia um ou mais arquivos armazenados no sistema de arquivo local para o sistema de arquivo remoto (ou vice-versa).

HTTP e FTP são protocolos de transferência de arquivos e têm muitas características em comum; por exemplo, ambos utilizam o TCP. Contudo, esses dois protocolos de camada de aplicação têm algumas diferenças importantes. A mais notável é que o FTP usa duas conexões TCP paralelas para transferir um arquivo: uma **conexão de controle** e uma **conexão de dados**. A primeira é usada para enviar informações de controle entre os dois hospedeiros — como identificação de usuário, senha, comandos para trocar diretório remoto e comandos de "enviar" (*put*) e "receber" (*get*) arquivos. A conexão de dados é a usada para enviar de fato um arquivo. Como o FTP usa uma conexão de controle separada, dizemos que ele envia suas informações de controle **fora da banda**. O HTTP, como você deve recordar, envia linhas de cabeçalho de requisição e de resposta pela mesma conexão TCP que carrega o próprio arquivo transferido. Por essa razão, dizemos que o HTTP envia suas informações de controle **na banda**. Na próxima seção, veremos que o SMTP, o principal protocolo para correio eletrônico, também envia suas informações de controle na banda. As conexões de controle e de dados do FTP estão ilustradas na Figura 2.15.

FIGURA 2.14 FTP TRANSPORTA ARQUIVOS ENTRE SISTEMAS DE ARQUIVO LOCAL E REMOTO

FIGURA 2.15 CONEXÕES DE CONTROLE E DE DADOS

Quando um usuário inicia uma sessão FTP com um hospedeiro remoto, o lado cliente do FTP (usuário) inicia primeiro uma conexão TCP de controle com o lado servidor (hospedeiro remoto) na porta número 21 do servidor e envia por essa conexão de controle a identificação e a senha do usuário, além de comandos para mudar o diretório remoto. Quando o lado servidor recebe, pela conexão de controle, um comando para uma transferência de arquivo (de ou para o hospedeiro remoto), abre uma conexão TCP de dados para o lado cliente. O FTP envia exatamente um arquivo pela conexão de dados e em seguida fecha-a. Se, durante a mesma sessão, o usuário quiser transferir outro arquivo, o FTP abrirá outra conexão de dados. Assim, com FTP, a conexão de controle permanece aberta durante toda a sessão do usuário, mas uma nova conexão de dados é criada para cada arquivo transferido dentro de uma sessão (ou seja, a conexão de dados é não persistente).

Durante uma sessão, o servidor FTP deve manter informações de **estado** sobre o usuário. Em particular, o servidor deve associar a conexão de controle com uma conta de usuário específica e também deve monitorar o diretório corrente do usuário enquanto este passeia pela árvore do diretório remoto. Monitorar essas informações de estado para cada sessão de usuário em curso limita de modo significativo o número total de sessões que o FTP pode manter simultaneamente. Lembre-se de que o HTTP, por outro lado, é sem estado — não tem de monitorar o estado de nenhum usuário.

2.3.1 Camadas e respostas FTP

Encerraremos esta seção com uma breve discussão sobre alguns dos comandos mais comuns do FTP e suas respostas. Os comandos, do cliente para o servidor, e as respostas, do servidor para o cliente, são enviados por meio da conexão de controle no formato ASCII de 7 bits. Assim, tal como comandos HTTP, comandos FTP também podem ser lidos pelas pessoas. Para separar comandos sucessivos, um "*carriage return*" e um "*line feed*" encerram cada um. Um comando é constituído de quatro caracteres ASCII maiúsculos, alguns com argumentos opcionais. Alguns dos comandos mais comuns são descritos a seguir:

- `USER username`: usado para enviar identificação do usuário ao servidor.
- `PASS password`: usado para enviar a senha do usuário ao servidor.

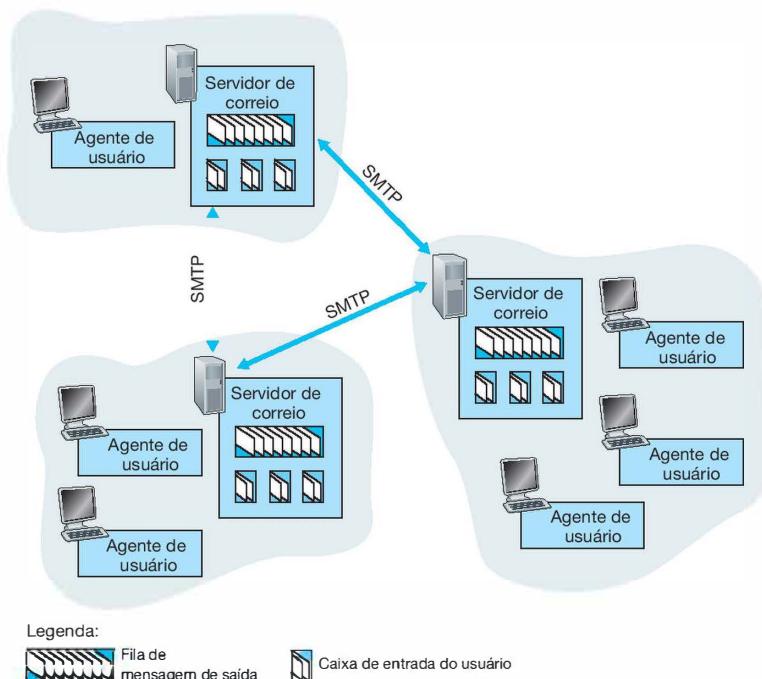
- `LIST:` usado para pedir ao servidor que envie uma lista com todos os arquivos existentes no atual diretório remoto. A lista de arquivos é enviada por meio de uma conexão de dados (nova e não persistente), e não pela conexão TCP de controle.
- `RETR filename:` usado para extrair (isto é, obter) um arquivo do diretório atual do hospedeiro remoto. Ativa o hospedeiro remoto para que abra uma conexão de dados e envia o arquivo requisitado por essa conexão.
- `STOR filename:` usado para armazenar (isto é, inserir) um arquivo no diretório atual do hospedeiro remoto.

Há, em particular, uma correspondência direta entre o comando que o usuário gera e o comando FTP enviado pela conexão de controle. Cada comando é seguido de uma resposta, que é enviada do servidor ao cliente. As respostas são números de três dígitos com uma mensagem opcional após o número. Elas se assemelham, em estrutura, ao código de estado e à frase da linha de estado da mensagem de resposta HTTP. Algumas respostas típicas, junto com suas possíveis mensagens, são as seguintes:

- `331 Nome de usuário OK, senha requisitada`
- `125 Conexão de dados já aberta; iniciando transferência`
- `425 Não é possível abrir a conexão de dados`
- `452 Erro ao escrever o arquivo`

Para saber mais sobre outros comandos e respostas FTP, o leitor interessado pode consultar o RFC 959.

2.4 CORREIO ELETRÔNICO NA INTERNET

O correio eletrônico existe desde o início da Internet. Era uma das aplicações mais populares quando ela ainda estava na infância [Segaller, 1998], e ficou mais e mais elaborado e poderoso ao longo dos anos. É uma das aplicações mais importantes e de maior uso na Internet.

Tal como o correio normal, o e-mail é um meio de comunicação assíncrono — as pessoas enviam e recebem mensagens quando for conveniente para elas, sem ter de estar coordenadas com o horário das outras. Ao contrário do correio normal, que anda a passos lentos, o eletrônico é rápido, fácil de distribuir e barato. O correio eletrônico moderno tem muitas características poderosas, incluindo mensagens com anexos, hiperlinks, textos formatados em HTML e fotos embutidas.

Nesta seção, examinaremos os protocolos de camada de aplicação que estão no cerne do correio eletrônico da Internet. Mas, antes de entrarmos nessa discussão, vamos tomar uma visão geral do sistema de correio da Internet e de seus componentes principais.

A Figura 2.16 apresenta uma visão do sistema de correio da Internet. Vemos, por esse diagrama, que há três componentes principais: **agentes de usuário**, **servidores de correio** e o **SMTP (Simple Mail Transfer Protocol)**. Descreveremos agora cada um deles no contexto de um remetente, Alice, enviando uma mensagem de e-mail para um destinatário, Bob. Agentes de usuários permitem que usuários leiam, respondam, encaminhem, salvem e componham mensagens. Microsoft Outlook e Apple Mail são alguns desses agentes. Quando Alice termina de compor sua mensagem, seu agente de usuário envia a mensagem para seu servidor de correio, onde a mensagem é colocada em sua fila de mensagens de saída. Quando Bob quer ler uma mensagem, seu agente de usuário apanha a mensagem de sua caixa de correio, em seu servidor de correio.

Servidores de correio formam o núcleo da infraestrutura do e-mail. Cada destinatário, como Bob, tem uma **caixa postal** localizada em um desses servidores. A de Bob administra e guarda as mensagens que foram enviadas a ele. Uma mensagem típica inicia sua jornada no agente de usuário do remetente, vai até seu servidor de correio e viaja até o do destinatário, onde é depositada na caixa postal. Quando Bob quer acessar as mensagens de sua caixa postal, o servidor de correio que contém sua caixa postal o autentica (com nome de usuário e senha). O servidor de correio de Alice também deve cuidar das falhas no servidor de correio de Bob. Se o servidor de correio dela não puder entregar a correspondência ao dele, manterá a mensagem em uma **fila**

FIGURA 2.16 UMA VISÃO DO SISTEMA DE E-MAIL DA INTERNET

de mensagens e tentará transferi-la mais tarde. Em geral, novas tentativas serão feitas a cada 30 minutos mais ou menos; se não obtiver sucesso após alguns dias, o servidor removerá a mensagem e notificará o remetente (Alice) por meio de uma mensagem de correio.

O SMTP é o principal protocolo de camada de aplicação do correio eletrônico da Internet. Usa o serviço confiável de transferência de dados do TCP para transferir mensagens do servidor de correio do remetente para o do destinatário. Como acontece com a maioria dos protocolos de camada de aplicação, o SMTP tem dois lados: um lado cliente, que funciona no servidor de correio do remetente, e um lado servidor, que funciona no servidor de correio do destinatário. Ambos funcionam em todos os servidores de correio. Quando um servidor de correio envia correspondência para outros, age como um cliente SMTP. Quando o servidor de correio recebe correspondência de outros, age como um servidor SMTP.

2.4.1 SMTP

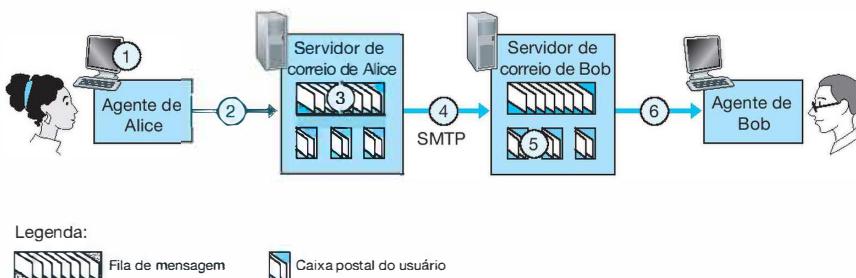
O SMTP, definido no RFC 5321, está no cerne do correio eletrônico da Internet. Como já dissemos, esse protocolo transfere mensagens de servidores de correio remetentes para servidores de correio destinatários. O SMTP é muito mais antigo que o HTTP. (O RFC original do SMTP data de 1982, e ele já existia muito antes disso.) Embora tenha inúmeras qualidades maravilhosas, como evidencia sua ubiquidade na Internet, o SMTP é uma tecnologia antiga que possui certas características arcaicas. Por exemplo, restringe o corpo (e não apenas o cabeçalho) de todas as mensagens de correio ao simples formato ASCII de 7 bits. Essa restrição tinha sentido no começo da década de 1980, quando a capacidade de transmissão era escassa e ninguém enviava correio eletrônico com anexos volumosos nem arquivos grandes com imagens, áudio ou vídeo. Mas, hoje, na era da multimídia, a

restrição do ASCII de 7 bits é um tanto incômoda — exige que os dados binários de multimídia sejam codificados em ASCII antes de ser enviados pelo SMTP e que a mensagem correspondente em ASCII seja decodificada novamente para o sistema binário depois do transporte pelo SMTP. Lembre-se da Seção 2.2, na qual dissemos que o HTTP não exige que os dados de multimídia sejam codificados em ASCII antes da transferência.

Para ilustrar essa operação básica do SMTP, vamos percorrer um cenário comum. Suponha que Alice queira enviar a Bob uma simples mensagem ASCII.

1. Alice chama seu agente de usuário para e-mail, fornece o endereço de Bob (por exemplo, `bob@someschool.edu`), compõe uma mensagem e instrui o agente de usuário a enviá-la.
2. O agente de usuário de Alice envia a mensagem para seu servidor de correio, onde ela é colocada em uma fila de mensagens.
3. O lado cliente do SMTP, que funciona no servidor de correio de Alice, vê a mensagem na fila e abre uma conexão TCP para um servidor SMTP, que funciona no servidor de correio de Bob.
4. Após alguns procedimentos iniciais de apresentação (*handshaking*), o cliente SMTP envia a mensagem de Alice pela conexão TCP.
5. No servidor de correio de Bob, o lado servidor do SMTP recebe a mensagem e a coloca na caixa postal dele.
6. Bob chama seu agente de usuário para ler a mensagem quando for mais conveniente para ele.

Esse cenário está resumido na Figura 2.17.

É importante observar que o SMTP em geral não usa servidores de correio intermediários para enviar correspondência, mesmo quando os dois servidores estão localizados em lados opostos do mundo. Se o servidor de Alice está em Hong Kong, e o de Bob, em St. Louis, a conexão TCP é uma conexão direta entre os servidores em Hong Kong e St. Louis. Em particular, se o servidor de correio de Bob não estiver em funcionamento, a mensagem permanece no de Alice esperando por uma nova tentativa — a mensagem não é colocada em nenhum servidor de correio intermediário.

HISTÓRIA

E-mail da Web

Em dezembro de 1995, apenas alguns anos depois da "invenção" da Web, Sabeer Bhatia e Jack Smith fizeram uma visita a Draper Fisher Jurvetson, um investidor em empreendimentos de Internet, e propuseram desenvolver um sistema de e-mail de livre acesso baseado na Web. A ideia era oferecer uma conta de e-mail grátis a quem quisesse e tornar tais contas acessíveis pela Web. Em troca de 15% da empresa, Draper Fisher Jurvetson financiou Bhatia e Smith, que formaram uma empresa denominada Hotmail. Com três funcionários em tempo integral e mais 14 em tempo parcial, que trabalhavam em troca de opções de compra de ações da empresa, eles conseguiram desenvolver e lançar o serviço em julho de 1996. Um mês após o lançamento, a Hotmail tinha cem mil assinantes. Em dezembro de 1997, menos de 18 meses depois, a Hotmail, já com mais de 12 milhões de assinantes, foi adquirida pela Microsoft, ao que se saiba, por 400 milhões de dólares. O sucesso da Hotmail é muitas vezes atribuído à vantagem de ela ter sido a primeira a entrar no mercado e ao inerente "marketing viral" do e-mail. (Talvez alguns dos estudantes que leem este livro estarão entre os novos empreendedores que conceberão e desenvolverão serviços de Internet pioneiros no mercado e com marketing viral.)

O e-mail da Web continua a prosperar, tornando-se, a cada ano, mais sofisticado e potente. Um dos serviços mais populares de hoje é o gmail do Google, que oferece livre armazenagem de gigabytes, filtro de *spam* e detector de vírus avançados, codificação de e-mail opcional (utilizando SSL), busca de correio de serviços de terceiros e uma interface orientada a busca. Serviços de mensagens assíncronas dentro de redes sociais, como Facebook, também se tornaram populares nos últimos anos.

FIGURA 2.17 ALICE ENVIA UMA MENSAGEM A BOB

Vamos agora examinar mais de perto como o SMTP transfere uma mensagem de um servidor de correio remetente para um servidor de correio destinatário. Veremos que o protocolo SMTP tem muitas semelhanças com protocolos usados na interação humana cara a cara. Primeiro, o cliente SMTP (que funciona no hospedeiro do servidor de correio remetente) faz o TCP estabelecer uma conexão na porta 25 com o servidor SMTP (que funciona no hospedeiro do servidor de correio destinatário). Se o servidor não estiver em funcionamento, o cliente tenta de novo mais tarde. Uma vez estabelecida a conexão, o servidor e o cliente trocam alguns procedimentos de apresentação de camada de aplicação — exatamente como os seres humanos, que costumam se apresentar antes de transferir informações, clientes e servidores SMTP também se apresentam antes de transferir informações. Durante essa fase, o cliente SMTP indica os endereços de e-mail do remetente (a pessoa que gerou a mensagem) e do destinatário. Assim que o cliente e o servidor SMTP terminam de se apresentar, o cliente envia a mensagem. O SMTP pode contar com o serviço confiável de transferência de dados do TCP para entregar a mensagem ao servidor sem erros. Então, o cliente repetirá esse processo, na mesma conexão TCP, se houver outras mensagens a enviar ao servidor; caso contrário, dará uma instrução ao TCP para encerrar a conexão.

Vamos analisar um exemplo de troca de mensagens entre um cliente (C) e um servidor SMTP (S). O nome do hospedeiro do cliente é `crepes.fr` e o nome do hospedeiro do servidor é `hamburger.edu`. As linhas de texto ASCII iniciadas com C: são exatamente as linhas que o cliente envia para dentro de seu *socket* TCP e as iniciadas com S: são exatamente as linhas que o servidor envia para dentro de seu *socket* TCP. A transcrição a seguir começa assim que a conexão TCP é estabelecida:

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr ... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Neste exemplo, o cliente enviou uma mensagem (“`Do you like ketchup? How about pickles?`”) do servidor de correio crepes.fr ao servidor de correio `hamburger.edu`. Como parte do diálogo, o cliente emitiu cinco comandos: HELO (uma abreviação de HELLO), `MAIL FROM`, `RCPT TO`, `DATA` e `QUIT`. Esses comandos são autoexplicativos. O cliente também enviou uma linha consistindo em um único ponto final, que indica o final da mensagem para o servidor. (No jargão ASCII, cada mensagem termina com `CRLF.CRLF`, em

que `CR` significa '*carriage return*' e `LF` significa '*line feed*'.) O servidor emite respostas a cada comando, e cada resposta tem uma codificação de resposta e algumas explicações (opcionais) em inglês. Mencionamos aqui que o SMTP usa conexões persistentes: se o servidor de correio remetente tiver diversas mensagens para enviar ao mesmo servidor de correio destinatário, poderá enviar todas pela mesma conexão TCP. Para cada mensagem, o cliente inicia o processo com um novo `MAIL FROM: crepes.fr`, indica o final da mensagem com um ponto final isolado e emite `QUIT` somente após todas as mensagens terem sido enviadas.

Recomendamos vivamente que você utilize o Telnet para executar um diálogo direto com um servidor SMTP. Para fazer isso digite

```
telnet serverName 25
```

em que serverName é o nome de um servidor de correio local. Ao fazer isso, você está apenas estabelecendo uma conexão TCP entre seu hospedeiro local e o servidor de correio. Após digitar essa linha, você deverá receber imediatamente do servidor a resposta 220. Digite, então, os comandos `HELO`, `MAIL FROM`, `RCPT TO`, `DATA`, `CRLF.CRLF` e `QUIT` nos momentos apropriados. Também recomendamos que você faça a Tarefa de Programação 2 no final deste capítulo. Nela, você construirá um agente de usuário simples que executa o lado cliente do SMTP. Esse agente permitirá que você envie uma mensagem de e-mail a um destinatário qualquer, por meio de um servidor de correio local.

2.4.2 Comparação com o HTTP

Agora, vamos fazer uma breve comparação entre o SMTP e o HTTP. Ambos os protocolos são usados para transferir arquivos de um hospedeiro para outro. O HTTP transfere arquivos (também denominados objetos) de um servidor para um cliente Web (em geral um navegador). O SMTP transfere arquivos (isto é, mensagens de e-mail) de um servidor de correio para outro. Ao transferir os arquivos, o HTTP persistente e o SMTP usam conexões persistentes. Assim, os dois protocolos têm características em comum. Existem, todavia, diferenças importantes. A primeira é que o HTTP é, principalmente, um **protocolo de recuperação** de informações (***pull protocol***) — alguém carrega informações em um servidor Web e os usuários utilizam o HTTP para recuperá-las quando quiserem. Em particular, a conexão TCP é ativada pela máquina que quer receber o arquivo. O SMTP, por sua vez, é, primordialmente, um **protocolo de envio** de informações (***push protocol***) — o servidor de correio remetente envia o arquivo para o servidor de correio destinatário. Em particular, a conexão TCP é ativada pela máquina que quer enviar o arquivo.

A segunda diferença, à qual aludimos anteriormente, é que o SMTP exige que cada mensagem, inclusive o corpo, esteja no formato ASCII de 7 bits. Se ela contiver caracteres que não estejam nesse formato (por exemplo, caracteres em francês, com acento) ou dados binários (como um arquivo de imagem), terá de ser codificada em ASCII de 7 bits. Dados HTTP não impõem esta restrição.

A terceira diferença importante refere-se ao modo como um documento que contém texto e imagem (juntamente com outros tipos possíveis de mídia) é manipulado. Como vimos na Seção 2.2, o HTTP encapsula cada objeto em sua própria mensagem HTTP. O correio pela Internet, como discutiremos com maiores detalhes mais adiante, coloca todos os objetos de mensagem em uma única mensagem.

2.4.3 Formatos de mensagem de correio

Quando Alice escreve uma carta a Bob e a envia pelo correio normal, ela pode incluir todos os tipos de informações periféricas no cabeçalho da carta, como seu próprio endereço, o endereço de Bob e a data. De modo semelhante, quando uma mensagem de e-mail é enviada, um cabeçalho contendo informações periféricas antecede o corpo da mensagem em si. Essas informações periféricas estão contidas em uma série de linhas de cabeçalho definidas no RFC 5322. As linhas de cabeçalho e o corpo da mensagem são separados por uma linha

em branco (isto é, por `CRLF`). O RFC 5322 especifica o formato exato das linhas de cabeçalho das mensagens, bem como suas interpretações semânticas. Como acontece com o HTTP, cada linha de cabeçalho contém um texto legível, consistindo em uma palavra-chave seguida de dois-pontos e de um valor. Algumas palavras-chave são obrigatórias e outras, opcionais. Cada cabeçalho deve ter uma linha de cabeçalho `From:` e uma `To:` e pode incluir também uma `Subject:` bem como outras opcionais. É importante notar que essas linhas de cabeçalho são *diferentes* dos comandos SMTP que estudamos na Seção 2.4.1 (ainda que contenham algumas palavras em comum, como '*from*' e '*to*'). Os comandos daquela seção faziam parte do protocolo de apresentação SMTP; as linhas de cabeçalho examinadas nesta seção fazem parte da própria mensagem de correio.

Um cabeçalho de mensagem típico é semelhante a:

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Searching for the meaning of life.
```

Após o cabeçalho da mensagem, vem uma linha em branco e, em seguida, o corpo da mensagem (em ASCII). Você pode usar o Telnet para enviar a um servidor de correio uma mensagem que contenha algumas linhas de cabeçalho, inclusive `Subject:`. Para tal, utilize o comando `telnet serverName 25`, como discutido na Seção 2.4.1.

2.4.4 Protocolos de acesso ao correio

Quando o SMTP entrega a mensagem do servidor de correio de Alice ao de Bob, ela é colocada na caixa postal de Bob. Durante toda essa discussão, ficou tacitamente subentendido que Bob lê sua correspondência ao entrar no hospedeiro servidor e, em seguida, executa o leitor de correio que roda naquela máquina. Até o início da década de 1990, este era o modo padronizado de acessar o correio. Mas hoje o acesso ao correio usa uma arquitetura cliente-servidor — o usuário típico lê e-mails com um cliente que funciona em seu sistema final, por exemplo, um PC no escritório, um laptop ou um smartphone. Quando executam um cliente de correio em PC local, usuários desfrutam de uma série de propriedades, inclusive a capacidade de ver mensagens e anexos multimídia.

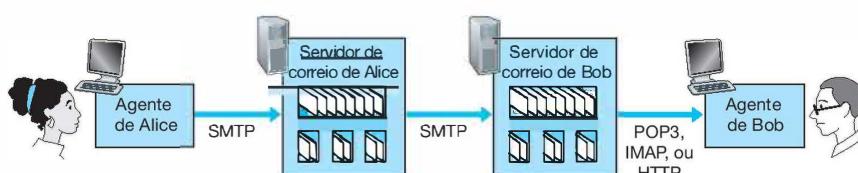
Dado que Bob (o destinatário) executa seu agente de usuário em seu PC local, é natural que ele considere a instalação de um servidor de correio também em seu PC local. Adotando essa abordagem, o servidor de correio de Alice dialogaria diretamente com o PC de Bob. Porém, há um problema com essa abordagem. Lembre-se de que um servidor de correio gerencia caixas postais e executa os lados cliente e servidor do SMTP. Se o servidor de correio de Bob residisse em seu PC local, este teria de ficar sempre em funcionamento e ligado na Internet para poder receber novas correspondências que poderiam chegar a qualquer hora, o que não é prático para muitos usuários. Em vez disso, um usuário típico executa um agente de usuário no PC local, mas acessa sua caixa postal armazenada em um servidor de correio compartilhado que está sempre em funcionamento. Esse servidor de correio é compartilhado com outros usuários e, em geral, é mantido pelo ISP do usuário (por exemplo, universidade ou empresa).

Agora, vamos considerar o caminho que uma mensagem percorre quando é enviada de Alice para Bob. Acabamos de aprender que, em algum ponto do percurso, a mensagem de e-mail precisa ser depositada no servidor de correio de Bob. Essa tarefa poderia ser realizada simplesmente fazendo o agente de usuário de Alice enviar a mensagem diretamente ao servidor de correio de Bob, o que pode ser feito com o SMTP — de fato, o SMTP foi projetado para enviar e-mail de um hospedeiro para outro. Contudo, em geral o agente de usuário do remetente não dialoga diretamente com o servidor de correio do destinatário. Em vez disso, como mostra a Figura 2.18, o agente de usuário de Alice usa SMTP para enviar a mensagem de e-mail a seu servidor de correio. Em seguida, esse servidor usa SMTP (como um cliente SMTP) para retransmitir a mensagem de e-mail ao servidor de correio de Bob. Por que esse procedimento em duas etapas? Primordialmente porque, sem a retransmissão pelo servidor de correio de Alice, o agente de usuário dela não dispõe de nenhum recurso para um servidor de correio de destinatário que não pode ser alcançado. Fazendo que Alice primeiro deposite o e-mail em seu próprio servidor de correio, este pode tentar, várias vezes, enviar a mensagem ao servidor de correio de Bob, digamos, a cada 30 minutos, até que esse servidor entre em operação. (E, se o servidor de correio de Alice não estiver funcionando, ela terá o recurso

de se queixar ao administrador do seu sistema!) O RFC do SMTP define como os comandos do SMTP podem ser usados para retransmitir uma mensagem por vários servidores SMTP.

Mas ainda falta uma peça do quebra-cabeça! De que forma um destinatário como Bob, que executa um agente de usuário em seu PC local, obtém suas mensagens que estão em um servidor de correio dentro do seu ISP? Note que o agente de usuário de Bob não pode usar SMTP para obter as mensagens porque essa operação é de recuperação (*pull*), e o SMTP é um protocolo de envio (*push*). O quebra-cabeça é concluído com a introdução de um protocolo especial de acesso ao correio que transfere mensagens do servidor de correio de Bob para seu PC local. Hoje, há vários protocolos populares de acesso a correio, entre eles **POP3 (Post Office Protocol versão 3)**, **IMAP (Internet Mail Access Protocol)** e HTTP.

A Figura 2.18 apresenta um resumo dos protocolos usados no correio da Internet: o SMTP é utilizado para transferir correspondência do servidor de correio remetente para o servidor de correio destinatário; também para transferir correspondência do agente de usuário remetente para o servidor de correio remetente. Um protocolo de acesso de correio, como o POP3, é utilizado para transferir correspondência do servidor de correio destinatário para o agente de usuário destinatário.

POP3

O POP3 é um protocolo de acesso de correio de extrema simplicidade. É definido no [RFC 1939], que é curto e bem fácil de ler. Por ser tão simples, sua funcionalidade é bastante limitada. O POP3 começa quando o agente de usuário (o cliente) abre uma conexão TCP com o servidor de correio (o servidor) na porta 110. Com a conexão TCP ativada, o protocolo passa por três fases: autorização, transação e atualização. Durante a primeira fase, autorização, o agente de usuário envia um nome de usuário e uma senha (às claras) para autenticar o usuário. Na segunda fase, transação, recupera mensagens; é também nessa etapa que o agente pode marcar mensagens que devem ser apagadas, remover essas marcas e obter estatísticas de correio. A terceira, atualização, ocorre após o cliente ter dado o comando quit que encerra a sessão POP3. Nesse momento, o servidor de correio apaga as mensagens que foram marcadas.

Em uma transação POP3, o agente de usuário emite comandos e o servidor, uma resposta para cada um deles. Há duas respostas possíveis: +OK (às vezes seguida de dados do servidor para o cliente), usada pelo servidor para indicar que correu tudo bem com o comando anterior e `-ERR`, que o servidor usa para informar que houve algo errado com o comando anterior.

A fase de autorização tem dois comandos principais: `user <username>` e `pass <password>`. Para ilustrá-los, sugerimos que você realize uma sessão Telnet diretamente com um servidor POP3, usando a porta 110, e emita os dois comandos. Suponha que `mailServer` seja o nome de seu servidor de correio. O que você verá será algo parecido com:

```
telnet mailServer 110
+OK POP3 server ready
user bob
+OK
pass hungry
+OK user successfully logged on
```

FIGURA 2.18 **PROTOCOLOS DE E-MAIL E SUAS ENTIDADES COMUNICANTES**

Se você escrever um comando errado, o POP3 responderá com uma mensagem `-ERR`.

Agora, vamos examinar a fase de transação. Um agente de usuário que utiliza POP3 com frequência pode ser configurado (pelo usuário) para "ler-e-apagar" ou para "ler-e-guardar". A sequência de comandos emitida por um agente de usuário POP3 depende do modo em que o agente de usuário estiver operando. No modo ler-e-apagar, o agente de usuário emite os comandos `list`, `retr` e `dele`. Como exemplo, suponha que o usuário tenha duas mensagens em sua caixa postal. No diálogo abaixo, `C:` (que representa o cliente) é o agente de usuário e `S:` (que representa o servidor), o servidor de correio. A transação será mais ou menos assim:

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: (blah blah ...
S: .................
S: ..........blah)
S: .
C: dele 1
C: retr 2
S: (blah blah ...
S: .................
S: ..........blah)
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

O agente de usuário primeiro pede ao servidor de correio que apresente o tamanho de cada mensagem armazenada. Então, recupera e apaga cada mensagem do servidor. Note que, após a fase de autorização, o agente de usuário empregou apenas quatro comandos: `list`, `retr`, `dele` e `quit`. A sintaxe para eles é definida no RFC 1939. Depois de processar o comando de saída (*quit*), o servidor POP3 entra na fase de atualização e remove as mensagens 1 e 2 da caixa postal.

Há um problema com o modo ler-e-apagar: o destinatário, Bob, pode ser nômade e querer acessar seu correio de muitas máquinas, por exemplo, do PC de seu escritório, do PC de sua casa e de seu computador portátil. O modo ler-e-apagar reparte as mensagens de correio de Bob entre essas três máquinas; em particular, se ele ler primeiro uma mensagem no PC de seu escritório, não poderá lê-la de novo mais tarde em seu computador portátil. No modo ler-e-guardar, o agente de usuário deixa as mensagens no servidor de correio após descarregá-las. Nesse caso, Bob pode reler mensagens em máquinas diferentes; pode acessar uma mensagem em seu local de trabalho e, uma semana depois, acessá-la novamente em casa.

Durante uma sessão POP3 entre um agente de usuário e o servidor de correio, o servidor POP3 mantém alguma informação de estado; em particular, monitora as mensagens do usuá–rio marcadas para apagar. Contudo, não mantém informação de estado entre sessões POP3. Essa falta de informação simplifica a execução de um servidor POP3.

IMAP

Usando POP3, assim que baixar suas mensagens na máquina local, Bob pode criar pastas de correspondência e transferir as mensagens baixadas para elas. Em seguida, consegue apagar as mensagens, mudá-las de pastas e procurar mensagens (por nome de remetente ou assunto). Mas esse paradigma — pastas e mensagens na máquina local — apresenta um problema para o usuário nômade que gostaria de manter uma hierarquia de pastas em um servidor remoto que possa ser acessado de qualquer computador: com o POP3, isso não é possível. Esse protocolo não provê nenhum meio para um usuário criar pastas remotas e designar mensagens a pastas.

Para resolver esse e outros problemas, foi inventado o protocolo IMAP, definido no [RFC 3501]. Como o POP3, o IMAP é um protocolo de acesso a correio, porém com mais recursos, mas é também significativamente mais complexo. (E, portanto, também as implementações dos lados cliente e servidor são muito mais complexas.)

Um servidor IMAP associa cada mensagem a uma pasta. Quando uma mensagem chega a um servidor pela primeira vez, é associada com a pasta INBOX do destinatário, que, então, pode transferi-la para uma nova pasta criada por ele, lê-la, apagá-la e assim por diante. O protocolo IMAP provê comandos que permitem aos usuários criarem pastas e transferir mensagens de uma para outra. Também provê comandos que os usuários podem usar para pesquisar pastas remotas em busca de mensagens que obedeçam a critérios específicos. Note que, ao contrário do POP3, um servidor IMAP mantém informação de estado de usuário entre sessões IMAP — por exemplo, os nomes das pastas e quais mensagens estão associadas a elas.

Outra característica importante do IMAP é que ele tem comandos que permitem que um agente de usuário obtenha componentes de mensagens. Por exemplo, um agente de usuário pode obter apenas o cabeçalho ou somente uma das partes de uma mensagem MIME multiparte. Essa característica é útil quando há uma conexão de largura de banda estreita (por exemplo, uma conexão de modem em baixa velocidade) entre o agente de usuário e seu servidor de correio. Com uma conexão de pequena largura de banda, o usuário pode decidir não baixar todas as mensagens de sua caixa postal, evitando, em particular, mensagens longas que possam conter, por exemplo, um clipe de áudio ou de vídeo.

E-mail pela Web

Hoje, um número cada vez maior de usuários está enviando e acessando e-mails por meio de seus navegadores Web. O Hotmail lançou o e-mail com acesso pela Web em meados da década de 1990; agora, esse tipo de acesso também é fornecido por Google, Yahoo!, bem como universidades e empresas importantes. Com esse serviço, o agente de usuário é um navegador Web comum e o usuário se comunica com sua caixa postal remota via HTTP. Quando um destinatário, por exemplo, Bob, quer acessar uma mensagem em sua caixa postal, ela é enviada do servidor de correio para o navegador de Bob usando o protocolo HTTP, e não os protocolos POP3 ou IMAP. Quando um remetente, por exemplo, Alice, quer enviar uma mensagem de e-mail, esta é enviada do navegador de Alice para seu servidor de correio por HTTP, e não por SMTP. O servidor de correio de Alice, contudo, ainda envia mensagens para outros servidores de correio e recebe mensagens de outros servidores de correio usando o SMTP.

2.5 DNS: O SERVIÇO DE DIRETÓRIO DA INTERNET

Nós, seres humanos, podemos ser identificados por diversas maneiras. Por exemplo, podemos ser identificados pelo nome que aparece em nossa certidão de nascimento, pelo número do RG ou da carteira de motorista. Embora cada um desses números possa ser usado para identificar pessoas, em um dado contexto um pode ser mais adequado que outro. Por exemplo, os computadores da Receita Federal preferem usar o número do CPF (de tamanho fixo) ao nome que consta em nossa certidão de nascimento. Por outro lado, pessoas comuns preferem nosso nome de batismo, mais fácil de lembrar, ao número do CPF. (Realmente, você pode se imaginar dizendo: "Oi, meu nome é 132-67-9875. Este é meu marido, 178-87-1146"?)

Assim como seres humanos podem ser identificados de muitas maneiras, o mesmo acontece com hospedeiros da Internet. Um identificador é seu **nome de hospedeiro** (*hostname*). Nomes de hospedeiro — como `cnn.com`, `www.yahoo.com`, `gaia.cs.umass.edu` e `cis.poly.edu` — são fáceis de lembrar e, portanto, apreciados pelos seres humanos. Todavia, eles fornecem pouca — se é que alguma — informação sobre a localização de um hospedeiro na Internet. (Um nome como `www.eurecom.fr`, que termina com o código do país `.fr`, nos informa que o hospedeiro deve estar na França, mas não diz muito mais do que isso.) Além disso, como nomes de hospedeiros podem consistir em caracteres alfanuméricos de comprimento variável, seriam difíceis de ser processados por roteadores. Por essas razões, hospedeiros também são identificados pelo que denominamos **endereços IP**.

Discutiremos endereços IP mais detalhadamente no Capítulo 4, mas é importante falar um pouco sobre eles agora. Um endereço IP é constituído de 4 bytes e sua estrutura hierárquica é rígida. Ele é semelhante a `121.7.106.83`, no qual cada ponto separa um dos bytes expressos em notação decimal de 0 a 255. Um endereço IP é hierárquico porque, ao examiná-lo da esquerda para a direita, obtemos gradativamente mais informações específicas sobre onde o hospedeiro está localizado na Internet (isto é, em qual rede, dentre as muitas que compõem a Internet). De maneira semelhante, quando examinamos um endereço postal de cima para baixo, obtemos informações cada vez mais específicas sobre a localização do destinatário.

2.5.1 Serviços fornecidos pelo DNS

Acabamos de ver que há duas maneiras de identificar um hospedeiro — por um nome de hospedeiro e por um endereço IP. As pessoas preferem o identificador nome de hospedeiro por ser mais fácil de lembrar, ao passo que roteadores preferem endereços IP de comprimento fixo e estruturados hierarquicamente. Para conciliar essas preferências, é necessário um serviço de diretório que traduza nomes de hospedeiro para endereços IP. Esta é a tarefa principal do **DNS** (***domain name system*** — **sistema de nomes de domínio**) da Internet. O DNS é (1) um banco de dados distribuído executado em uma hierarquia de **servidores de DNS**, e (2) um protocolo de camada de aplicação que permite que hospedeiros consultem o banco de dados distribuído. Os servidores DNS são muitas vezes máquinas UNIX que executam o software BIND (Berkeley Internet Name Domain) [BIND, 2012]. O protocolo DNS utiliza UDP e usa a porta 53.

O DNS costuma ser empregado por outras entidades da camada de aplicação — inclusive HTTP, SMTP e FTP — para traduzir nomes de hospedeiros fornecidos por usuários para endereços IP. Como exemplo, considere o que acontece quando um navegador (isto é, um cliente HTTP), que executa na máquina de algum usuário, requisita o URL `www.someschool.edu/index.html`. Para que a máquina do usuário possa enviar uma mensagem de requisição HTTP ao servidor Web `www.someschool.edu`, ela precisa primeiro obter o endereço IP. Isso é feito da seguinte maneira:

1. A própria máquina do usuário executa o lado cliente da aplicação DNS.
2. O navegador extrai o nome de hospedeiro, `www.someschool.edu`, do URL e passa o nome para o lado cliente da aplicação DNS.
3. O cliente DNS envia uma consulta contendo o nome do hospedeiro para um servidor DNS.
4. O cliente DNS por fim recebe uma resposta, que inclui o endereço IP correspondente ao nome de hospedeiro.
5. Tão logo o navegador receba o endereço do DNS, pode abrir uma conexão TCP com o processo servidor HTTP localizado na porta 80 naquele endereço IP.

Vemos, por esse exemplo, que o DNS adiciona mais um atraso — às vezes substancial — às aplicações de Internet que o usam. Felizmente, como discutiremos mais adiante, o endereço IP procurado quase sempre está no *cache* de um servidor DNS "próximo", o que ajuda a reduzir o tráfego DNS na rede, bem como o atraso médio do DNS.

O DNS provê alguns outros serviços importantes além da tradução de nomes de hospedeiro para endereços IP:

- **Apelidos (*aliasing*) de hospedeiro.** Um hospedeiro com nome complicado pode ter um ou mais apelidos. Um nome como `relay1.west-coast.enterprise.com` pode ter, por exemplo, dois apelidos, como `enterprise.com` e `www.enterprise.com`. Nesse caso, o nome de hospedeiro `relay1.west-coast.enterprise.com` é denominado **nome canônico**. Apelidos, quando existem, são em geral mais fáceis de lembrar do que o nome canônico. O DNS pode ser chamado por uma aplicação para obter o nome canônico correspondente a um apelido fornecido, bem como para obter o endereço IP do hospedeiro.
- **Apelidos de servidor de correio.** Por razões óbvias, é adequado que endereços de e-mail sejam fáceis de lembrar. Por exemplo, se Bob tem uma conta no Hotmail, seu endereço de e-mail pode ser simplesmente `bob@hotmail.com`. Contudo, o nome de hospedeiro do servidor do Hotmail é mais complicado e

muito mais difícil de lembrar do que apenas hotmail.com (por exemplo, o nome canônico pode ser algo parecido com `relay1.west-coast.hotmail.com`). O DNS pode ser chamado por uma aplicação de correio para obter o nome canônico a partir de um apelido fornecido, bem como o endereço IP do hospedeiro. Na verdade, o registro MX (veja adiante) permite que o servidor de correio e o servidor Web de uma empresa tenham nomes (apelidos) idênticos; por exemplo, o servidor Web e o servidor de correio de uma empresa podem ambos ser denominados `enterprise.com`.

- **Distribuição de carga.** O DNS também é usado para realizar distribuição de carga entre servidores replicados, tais como os servidores Web replicados. Sites movimentados, como `cnn.com`, são replicados em vários servidores, cada qual rodando em um sistema final e com um endereço IP diferentes. Assim, no caso de servidores Web replicados, um *conjunto* de endereços IP fica associado a um único nome canônico e contido no banco de dados do DNS. Quando clientes consultam um nome mapeado para um conjunto de endereços, o DNS responde com o conjunto inteiro de endereços IP, mas faz um rodízio da ordem deles dentro de cada resposta. Como um cliente em geral envia sua mensagem de requisição HTTP ao endereço IP que ocupa o primeiro lugar no conjunto, o rodízio de DNS distribui o tráfego entre os servidores replicados. O rodízio de DNS também é usado para e-mail, de modo que vários servidores de correio podem ter o mesmo apelido. Há pouco, empresas distribuidoras de conteúdo como a Akamai passaram a usar o DNS de maneira mais sofisticada [Dilley, 2002] para prover distribuição de conteúdo na Web (veja Capítulo 7).

O DNS está especificado no RFC 1034 e no RFC 1035 e atualizado em diversos RFCs adicionais. É um sistema complexo e, neste livro, apenas mencionamos os aspectos fundamentais de sua operação. O leitor interessado pode consultar os RFCs citados, o livro escrito por Abitz e Liu [Abitz, 1993] e também o artigo de Mockapetris [1988], que apresenta uma retrospectiva e uma ótima descrição do que e do porquê do DNS, e Mockapetris [2005].

2.5.2 Visão geral do modo de funcionamento do DNS

Apresentaremos, agora, uma visão panorâmica do modo de funcionamento do DNS. Nossa discussão focalizará o serviço de tradução de nome de hospedeiro para endereço IP.

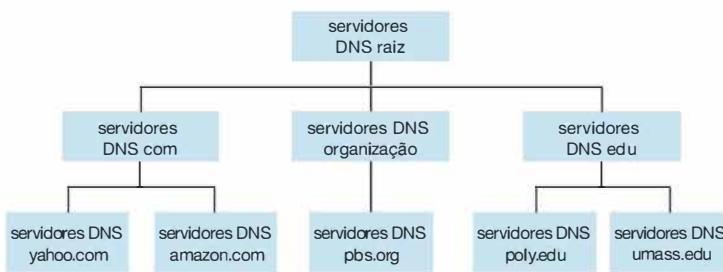
Suponha que certa aplicação (como um navegador Web ou um leitor de correio), que executa na máquina de um usuário, precise traduzir um nome de hospedeiro para um endereço IP. A aplicação chamará o lado cliente do DNS, especificando o nome de hospedeiro que precisa ser traduzido. (Em muitas máquinas UNIX, `gethostbyname()` é a chamada de função que uma aplicação invoca para realizar a tradução.) A partir daí, o DNS do hospedeiro do usuário assume o controle, enviando uma mensagem de consulta para a rede. Todas as mensagens de consulta e

PRINCÍPIOS NA PRÁTICA

Funções decisivas de rede via paradigma cliente-servidor

Como HTTP, FTP e SMTP, o DNS é um protocolo da camada de aplicação, já que (1) roda entre sistemas finais comunicantes usando o paradigma cliente-servidor e (2) depende de um protocolo de transporte fim a fim subjacente para transferir mensagens DNS entre sistemas finais comunicantes. Em outro sentido, contudo, o papel do DNS é bastante diferente das aplicações Web, da transferência de arquivo e do e-mail. Ao contrário delas, o DNS não é uma aplicação com a qual o usuário interage diretamente. Em vez disso, fornece uma função interna da Internet — a saber, a tradução de nomes de hospedeiro para seus endereços IP subjacentes, para aplicações de usuário e outros softwares da Internet. Notamos, na Seção 1.2, que grande parte da complexidade da arquitetura da Internet está localizada na “periferia” da rede. O DNS, que executa o processo crucial de tradução de nome para endereço usando clientes e servidores localizados nas bordas da rede, é mais um exemplo dessa filosofia de projeto.

de resposta do DNS são enviadas dentro de datagramas UDP à porta 53. Após um atraso na faixa de milissegundos a segundos, o DNS no hospedeiro do usuário recebe uma mensagem de resposta DNS fornecendo o mapeamento desejado, que é, então, passado para a aplicação que está interessada. Portanto, do ponto de vista dessa aplicação, que está na máquina do cliente, o DNS é uma caixa-preta que provê um serviço de tradução simples e direto. Mas, na realidade, a caixa-preta que executa o serviço é complexa, consistindo em um grande número de servidores DNS distribuídos ao redor do mundo, bem como em um protocolo de camada de aplicação que especifica como se comunicam os servidores DNS e os hospedeiros que fazem a consulta.

Um arranjo simples para DNS seria ter um servidor DNS contendo todos os mapeamentos. Nesse projeto centralizado, os clientes apenas dirigiriam todas as consultas a esse único servidor DNS, que responderia diretamente aos clientes que estão fazendo a consulta. Embora a simplicidade desse arranjo seja atraente, ele não é adequado para a Internet de hoje com seu vasto (e crescente) número de hospedeiros. Dentre os problemas de um arranjo centralizado, estão:

- **Um único ponto de falha.** Se o servidor DNS quebrar, a Internet inteira quebrará!
- **Volume de tráfego.** Um único servidor DNS teria de manipular todas as consultas DNS (para todas as requisições HTTP e mensagens de e-mail geradas por centenas de milhões de hospedeiros).
- **Banco de dados centralizado distante.** Um único servidor DNS nunca poderia estar "próximo" de todos os clientes que fazem consultas. Se colocarmos o único servidor DNS na cidade de Nova York, todas as buscas provenientes da Austrália terão de viajar até o outro lado do globo, talvez por linhas lentas e congestionadas, o que pode resultar em atrasos significativos.
- **Manutenção.** O único servidor DNS teria de manter registros de todos os hospedeiros da Internet. Esse banco de dados não só seria enorme, mas também precisaria ser atualizado frequentemente para atender a todos os novos hospedeiros.

Resumindo, um banco de dados centralizado em um único servidor DNS simplesmente não é escalável. Por conseguinte, o DNS é distribuído por conceito de projeto. Na verdade, ele é um ótimo exemplo de como um banco de dados distribuído pode ser executado na Internet.

Um banco de dados distribuído e hierárquico

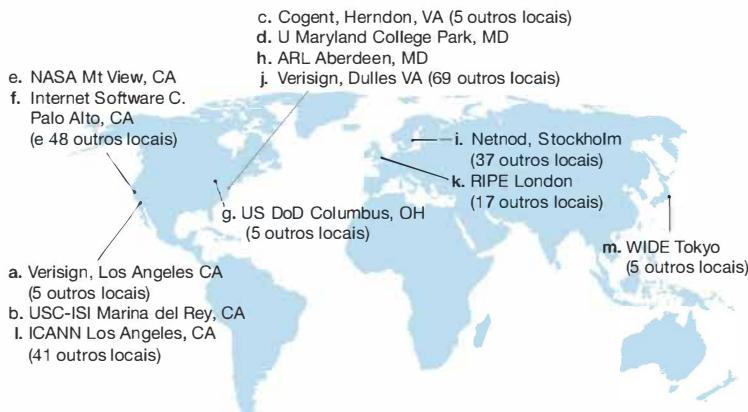
Para tratar da questão da escala, o DNS usa um grande número de servidores, organizados de maneira hierárquica e distribuídos por todo o mundo. Nenhum servidor DNS isolado tem todos os mapeamentos para todos os hospedeiros da Internet. Em vez disso, os mapeamentos são distribuídos pelos servidores DNS. Como uma primeira aproximação, há três classes de servidores DNS: raiz, de domínio de alto nível (*top-level domain* — TLD) e servidores DNS autoritativos — organizados em uma hierarquia, como mostra a Figura 2.19. Para entender como essas três classes interagem, suponha que um cliente DNS queira determinar o endereço IP para o nome de hospedeiro `www.amazon.com`. Como uma primeira aproximação, ocorrerão os seguintes eventos. Primeiro, o cliente contatará um dos servidores raiz, que retornará endereços IP dos servidores TLD para o domínio de alto nível com. Então, o cliente contatará um desses servidores TLD, que retornará o endereço IP de um servidor autoritativo para amazon.com. Por fim, o cliente contatará um dos servidores autoritativos para `amazon.com`, que retornará o endereço IP para o nome de hospedeiro `www.amazon.com`. Mais adiante, analisaremos em detalhes esse processo de consulta DNS. Mas, primeiro, vamos examinar mais de perto as três classes de servidores DNS:

- **Servidores DNS raiz.** Na Internet há 13 servidores DNS raiz (denominados de A a M) e a maior parte deles está localizada na América do Norte. Um mapa de servidores DNS raiz de outubro de 2006 é mostrado na Figura 2.20; uma lista dos servidores DNS raiz existentes hoje está disponível em Root-servers [2012]. Embora tenhamos nos referido a cada um dos 13 servidores DNS raiz como se fossem um servidor único, na realidade, cada um é um conglomerado de servidores replicados, para fins de segurança e confiabilidade. No total, havia 247 servidores raiz no segundo semestre de 2011.
- **Servidores DNS de Domínio de Alto Nível (TLD).** Esses servidores são responsáveis por domínios de alto nível como *com*, *org*, *net*, *edu* e *gov*, e por todos os domínios de alto nível de países, tais como *uk*, *fr*,

FIGURA 2.19 PARTE DA HIERARQUIA DE SERVIDORES DNS

ca e *jp*. A empresa Verisign Global Registry Services mantém os servidores TLD para o domínio de alto nível *com* e a Educause mantém os servidores TLD para o domínio de alto nível *edu*. Veja em IANA TLD [2012] uma lista de todos os domínios de alto nível.

- **Servidores DNS autoritativos.** Toda organização que tiver hospedeiros que possam ser acessados publicamente na Internet (como servidores Web e servidores de correio) deve fornecer registros DNS também acessíveis publicamente que mapeiem os nomes desses hospedeiros para endereços IP. Um servidor DNS autoritativo de uma organização abriga esses registros. Uma organização pode preferir executar seu próprio servidor DNS autoritativo para abrigar esses registros ou, como alternativa, pagar para armazená-los em um servidor DNS autoritativo de algum provedor de serviço. A maioria das universidades e empresas de grande porte executa e mantém seus próprios servidores DNS primário e secundário (*backup*) autoritativos.

Os servidores DNS raiz, TLD e autoritativo pertencem à hierarquia de servidores DNS, como mostra a Figura 2.19. Há mais um tipo importante de DNS, denominado **servidor DNS local**, que não pertence, estritamente, à hierarquia de servidores, mas, mesmo assim, é central para a arquitetura DNS. Cada ISP — como o de uma universidade, de um departamento acadêmico, de uma empresa ou de uma residência — tem um servidor DNS local (também denominado servidor DNS *default*). Quando um hospedeiro se conecta com um ISP, este fornece os endereços IP de um ou mais de seus servidores DNS locais (em geral por DHCP, que será discutido no Capítulo 4). Determinar o endereço IP do seu servidor DNS local é fácil: basta acessar as janelas de estado

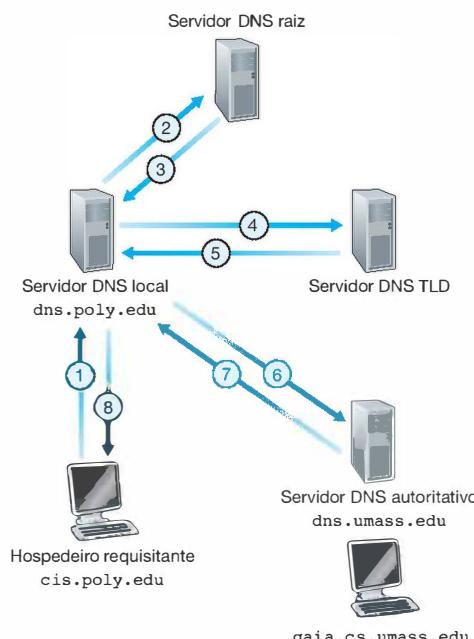
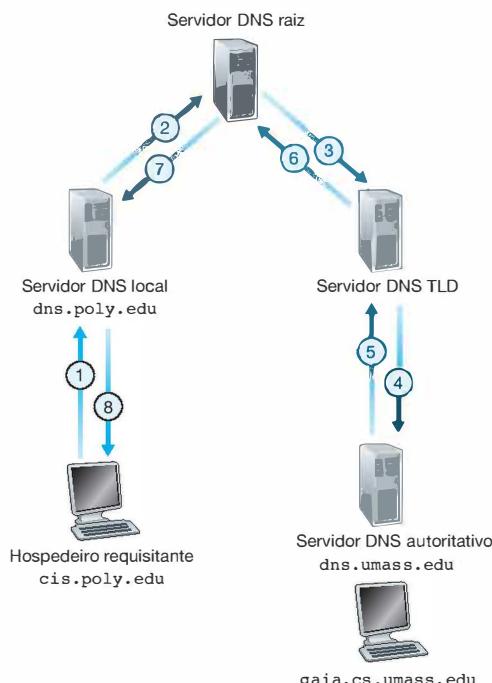



FIGURA 2.20 SERVIDORES DNS RAIZ EM 2012 (NOME, ORGANIZAÇÃO, LOCALIZAÇÃO)

da rede no Windows ou UNIX. O servidor DNS local de um hospedeiro costuma estar “próximo” dele. No caso de um ISP institucional, pode estar na mesma LAN do hospedeiro; já no caso de um ISP residencial, em geral o servidor DNS está separado do hospedeiro por não mais do que alguns roteadores. Quando um hospedeiro faz uma consulta ao DNS, ela é enviada ao servidor DNS local, que age como *proxy* e a retransmite para a hierarquia do servidor DNS, como discutiremos mais detalhadamente a seguir.

Vamos examinar um exemplo simples. Suponha que o hospedeiro `cis.poly.edu` deseje o endereço IP de `gaia.cs.umass.edu`. Imagine também que o servidor DNS local da Polytechnic seja denominado `dns.poly.edu` e que um servidor DNS autoritativo para `gaia.cs.umass.edu` seja denominado `dns.umass.edu`. Como mostra a Figura 2.21, o hospedeiro `cis.poly.edu` primeiro envia uma mensagem de consulta DNS a seu servidor DNS local `dns.poly.edu`. A mensagem de consulta contém o nome de hospedeiro a ser traduzido, isto é, `gaia.cs.umass.edu`. O servidor DNS local transmite a mensagem de consulta a um servidor DNS raiz, que percebe o sufixo edu e retorna ao servidor DNS local uma lista de endereços IP contendo servidores TLD responsáveis por edu. Então, o servidor DNS local retransmite a mensagem de consulta a um desses servidores TLD que, por sua vez, percebe o sufixo `umass.edu` e responde com o endereço IP do servidor DNS autorizado para a University of Massachusetts, a saber, dns.umass.edu. Por fim, o servidor DNS local reenvia a mensagem de consulta diretamente a `dns.umass.edu`, que responde com o endereço IP de `gaia.cs.umass.edu`. Note que, neste exemplo, para poder obter o mapeamento para um único nome de hospedeiro, foram enviadas oito mensagens DNS: quatro mensagens de consulta e quatro de resposta! Em breve veremos como o *cache* de DNS reduz esse tráfego de consultas.

Nosso exemplo anterior considerou que o servidor TLD conhece o servidor DNS autoritativo para o nome de hospedeiro, o que em geral não acontece. Ele pode conhecer apenas um servidor DNS intermediário que, por sua vez, conhece o servidor DNS autoritativo para o nome de hospedeiro. Por exemplo, suponha de novo que a Universidade de Massachusetts tenha um servidor DNS para a universidade denominado `dns.umass.edu`.

FIGURA 2.21 INTERAÇÃO DOS DIVERSOS SERVIDORES DNS

Imagine também que cada departamento da universidade tenha seu próprio servidor DNS e que cada servidor DNS departamental seja um servidor DNS autoritativo para todos os hospedeiros do departamento. Nesse caso, quando o servidor DNS intermediário dns.umass.edu receber uma consulta para um hospedeiro cujo nome termina com `cs.umass.edu`, ele retornará a `dns.poly.edu` o endereço IP de `dns.cs.umass.edu`, que tem autoridade para todos os nomes de hospedeiro que terminam com `cs.umass.edu`. Então, o servidor DNS local `dns.poly.edu` enviará a consulta ao servidor DNS autoritativo, que retornará o mapeamento desejado para o servidor DNS local e que, por sua vez, o repassará ao hospedeiro requisitante. Nesse caso, serão enviadas dez mensagens DNS no total!

O exemplo mostrado na Figura 2.21 usa **consultas recursivas** e **consultas iterativas**. A consulta enviada de `cis.poly.edu` para `dns.poly.edu` é recursiva, visto que pede a dns.poly.edu que obtenha o mapeamento em seu nome. Mas as três consultas subsequentes são iterativas, visto que todas as respostas são retornadas diretamente a `dns.poly.edu`. Em teoria, qualquer consulta DNS pode ser iterativa ou recursiva. Por exemplo, a Figura 2.22 mostra uma cadeia de consultas DNS na qual todas são recursivas. Na prática, as consultas em geral seguem o padrão mostrado na Figura 2.21: a consulta do hospedeiro requisitante ao servidor DNS local é recursiva e todas as outras são iterativas.

Cache DNS

Até aqui, nossa discussão ignorou o ***cache* DNS**, uma característica muito importante do sistema DNS. Na realidade, o DNS explora extensivamente o *cache* para melhorar o desempenho quanto ao atraso e reduzir o número de mensagens DNS que dispara pela Internet. A ideia por trás do *cache* DNS é muito simples. Em uma cadeia de consultas, quando um servidor DNS recebe uma resposta DNS (contendo, por exemplo, o mapea-

FIGURA 2.22 CONSULTAS RECURSIVAS EM DNS

mento de um nome de hospedeiro para um endereço IP), pode fazer *cache* das informações da resposta em sua memória local. Por exemplo, na Figura 2.21, toda vez que o servidor DNS local `dns.poly.edu` recebe uma resposta de algum servidor DNS, pode fazer *cache* de qualquer informação contida na resposta. Se um par nome de hospedeiro/endereço IP estiver no *cache* de um servidor DNS e outra consulta chegar ao mesmo servidor para o mesmo nome de hospedeiro, o servidor DNS poderá fornecer o endereço IP desejado, mesmo que não tenha autoridade para esse nome. Como hospedeiros e mapeamentos entre hospedeiros e endereços IP não são, de modo algum, permanentes, após um período de tempo (quase sempre dois dias), os servidores DNS descartam as informações armazenadas em seus *caches*.

Como exemplo, imagine que um hospedeiro `apricot.poly.edu` consulte `dns.poly.edu` para o endereço IP da máquina cnn.com. Além disso, suponha que algumas horas mais tarde outra máquina da Polytechnic University, digamos, `kiwi.poly.fr` também consulte dns.poly.edu para o mesmo nome de hospedeiro. Por causa do *cache*, o servidor local poderá imediatamente retornar o endereço IP de cnn.com a esse segundo hospedeiro requisitante, sem ter de consultar quaisquer outros servidores DNS. Um servidor DNS local também pode fazer *cache* de endereços IP de servidores TLD, permitindo, assim, que servidores DNS locais evitem os servidores DNS raiz em uma cadeia de consultas (isso acontece bastante).

2.5.3 Registros e mensagens DNS

Os servidores DNS que juntos executam o banco de dados distribuído do DNS armazenam **registros de recursos (RR)** que fornecem mapeamentos de nomes de hospedeiros para endereços IP. Cada mensagem de resposta DNS carrega um ou mais registros de recursos. Nesta seção e na subsequente, apresentaremos uma breve visão geral dos registros de recursos e mensagens DNS. Para mais detalhes, consulte Abitz [1993] ou RFC 1034; RFC 1035.

Um registro de recurso é uma tupla de quatro elementos que contém os seguintes campos:

```
(Name, Value, Type, TTL)
```

TTL é o tempo de vida útil do registro de recurso; determina quando um recurso deve ser removido de um *cache*. Nos exemplos de registros dados a seguir, ignoramos o campo TTL. Os significados de Name e Value dependem de Type:

- `Se Type=A`, então Name é um nome de hospedeiro e Value é o endereço IP para o nome de hospedeiro. Assim, um registro Type A fornece o mapeamento-padrão entre nomes de hospedeiros e endereços IP. Como exemplo, (`relay1.bar.foo.com`, `145.37.93.126`, A) é um registro com Type igual a A.
- `Se Type=NS`, então Name é um domínio (como `foo.com`) e Value é o nome de um servidor DNS autoritativo que sabe como obter os endereços IP para hospedeiros do domínio. Esse registro é usado para encaminhar consultas DNS ao longo da cadeia de consultas. Como exemplo, (`foo.com, dns.foo.com, NS`) é um registro com Type igual a NS.
- `Se Type=CNAME`, então Value é um nome canônico de hospedeiro para o apelido de hospedeiro contido em Name. Esse registro pode fornecer aos hospedeiros consultantes o nome canônico correspondente a um apelido de hospedeiro. Como exemplo, (`foo.com, relay1.bar.foo.com, CNAME`) é um registro CNAME.
- `Se Type=MX`, então Value é o nome canônico de um servidor de correio cujo apelido de hospedeiro está contido em Name. Como exemplo, (`foo.com, mail.bar.foo.com`, `MX`) é um registro MX. Registros MX permitem que os nomes de hospedeiros de servidores de correio tenham apelidos simples. Note que, usando o registro MX, uma empresa pode ter o mesmo apelido para seu servidor de arquivo e para um de seus outros servidores (tal como seu servidor Web). Para obter o nome canônico do servidor de correio, um cliente DNS consultaria um registro MX; para obter o nome canônico do outro servidor, o cliente DNS consultaria o registro CNAME.

Se um servidor DNS tiver autoridade para determinado nome de hospedeiro, então conterá um registro Type A para o nome de hospedeiro. (Mesmo que não tenha autoridade, o servidor DNS pode conter um registro Type A em seu *cache*.) Se um servidor não tiver autoridade para um nome de hospedeiro, conterá um registro Type NS para o domínio que inclui o nome e um registro Type A que fornece o endereço IP do servidor DNS no campo Value do registro NS. Como exemplo, suponha que um servidor TLD edu não tenha autoridade para o hospedeiro `gaia.cs.umass.edu`. Nesse caso, esse servidor conterá um registro para um domínio que inclui o hospedeiro `gaia.cs.umass.edu`, por exemplo (`umass.edu, dns.umass.edu, NS`). O servidor TLD edu conterá também um registro Type A, que mapeia o servidor DNS dns.umass.edu para um endereço IP, por exemplo (`dns.umass.edu, 128.119.40.111, A`).

Mensagens DNS

Abordamos anteriormente nesta seção mensagens de consulta e de resposta DNS, que são as duas únicas espécies de mensagem DNS. Além disso, tanto as mensagens de consulta como as de resposta têm o mesmo formato, como ilustra a Figura 2.23. A semântica dos vários campos de uma mensagem DNS é a seguinte:

- Os primeiros 12 bytes formam a *seção de cabeçalho*, que tem vários campos. O primeiro campo é um número de 16 bits que identifica a consulta. Esse identificador é copiado para a mensagem de resposta a uma consulta, permitindo que o cliente combine respostas recebidas com consultas enviadas. Há várias *flags* no campo de *flag*. Uma *flag* de consulta/resposta de 1 bit indica se a mensagem é uma consulta (0) ou uma resposta (1). Uma flag de autoridade de 1 bit é marcada em uma mensagem de resposta quando o servidor DNS é um servidor autoritativo para um nome consultado. Uma *flag* de recursão desejada de 1 bit é estabelecida quando um cliente (hospedeiro ou servidor DNS) quer que um servidor DNS proceda recursivamente sempre que não tem o registro. Um campo de recursão disponível de 1 bit é marcado em uma resposta se o servidor DNS suporta recursão. No cabeçalho, há também quatro campos de "tamanho". Eles indicam o número de ocorrências dos quatro tipos de seção de dados que se seguem ao cabeçalho.
- A *seção de pergunta* contém informações sobre a consulta que está sendo feita. Essa seção inclui (1) um campo de nome que contém o nome que está sendo consultado e (2) um campo de tipo que indica o tipo

FIGURA 2.23 FORMATO DA MENSAGEM DNS

da pergunta que está sendo feita sobre o nome — por exemplo, um endereço de hospedeiro associado a um nome (Type A) ou o servidor de correio para um nome (Type MX).

- Em uma resposta de um servidor DNS, a *seção de resposta* contém os registros de recursos para o nome que foi consultado originalmente. Lembre-se de que em cada registro de recurso há o `Type` (por exemplo, A, NS, CSNAME e MX), o Value e o TTL. Uma resposta pode retornar vários RRs, já que um nome de hospedeiro pode ter diversos endereços IP (por exemplo, para servidores Web replicados, como já discutimos anteriormente nesta seção).
- A *seção de autoridade* contém registros de outros servidores autoritativos.
- A *seção adicional* contém outros registros úteis. Por exemplo, o campo resposta em uma resposta a uma consulta MX conterá um registro de recurso que informa o nome canônico de um servidor de correio. A seção adicional conterá um registro Type A que fornece o endereço IP para o nome canônico do servidor de correio.

Você gostaria de enviar uma mensagem de consulta DNS direto de sua máquina a algum servidor DNS? Isso pode ser feito facilmente com o **programa nslookup**, que está disponível na maioria das plataformas Windows e UNIX. Por exemplo, se um hospedeiro executar Windows, abra o Prompt de comando e chame o programa nslookup apenas digitando "nslookup". Depois de chamar o programa, você pode enviar uma consulta DNS a qualquer servidor DNS (raiz, TLD ou autoritativo). Após receber a mensagem de resposta do servidor DNS, o nslookup apresentará os registros incluídos na resposta (em formato que pode ser lido normalmente). Como alternativa para executar nslookup na sua própria máquina, você pode visitar um dos muitos sites que permitem o emprego remoto do programa. (Basta digitar "nslookup" em um buscador e você será levado a um desses sites.) O Wireshark lab DNS, ao final deste capítulo, lhe permitirá explorar o DNS com muito mais detalhes.

Inserindo registros no banco de dados do DNS

A discussão anterior focalizou como são extraídos registros do banco de dados DNS. É possível que você esteja se perguntando como os registros entraram no banco de dados em primeiro lugar. Vamos examinar como isso é feito no contexto de um exemplo específico. Imagine que você acabou de criar uma nova empresa muito interessante denominada Network Utopia. A primeira coisa que você certamente deverá fazer é registrar o nome de domínio `networkutopia.com` em uma entidade registradora. Uma **entidade registradora** é uma organização comercial que verifica se o nome de domínio é exclusivo, registra-o no banco de dados do DNS (como discutiremos mais adiante) e cobra uma pequena taxa por seus serviços. Antes de 1999, uma única entidade registradora, a Network Solutions, detinha o monopólio do registro de nomes para os domínios `com`, `net` e `org`. Mas agora existem muitas entidades registradoras credenciadas pela Internet Corporation for Assigned Names and Numbers (ICANN) competindo por clientes. Uma lista completa dessas entidades está disponível em `http://www.internic.net`.

Ao registrar o nome de domínio networkutopia.com, você também precisará informar os nomes e endereços IP dos seus servidores DNS com autoridade, primários e secundários. Suponha que os nomes e endereços IP sejam `dns1.networkutopia.com`, `dns2.networkutopia.com`, `212.212.212.1` e `212.212.212.2`. A entidade registradora ficará encarregada de providenciar a inserção dos registros Type NS e Type A nos servidores TLD do domínio *com* para cada um desses dois servidores de nomes autorizados. Em especial, para o servidor primário com autoridade `networkutopia.com`, a autoridade registradora inseriria no sistema DNS os dois registros de recursos a seguir:

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

Não se esqueça de providenciar também a inserção em seus servidores de nomes com autoridade do registro de recurso Type A para seu servidor Web `www.networkutopia.com` e o registro de recurso Type MX para seu servidor de correio `mail.networkutopia.com`. (Até há pouco tempo, o conteúdo de cada servidor DNS era configurado estaticamente, por exemplo, a partir de um arquivo de configuração criado por um gerenciador

SEGURANÇA EM FOCO

Vulnerabilidades do DNS

Vimos que o DNS é um componente fundamental da infraestrutura da Internet, com muitos serviços importantes — incluindo a Web e o e-mail — simplesmente incapazes de funcionar sem ele. Dessa maneira, perguntamos: como o DNS pode ser atacado? O DNS é um alvo esperando para ser atingido, pois causa dano à maioria das aplicações da Internet junto com ele?

O primeiro tipo de ataque que vem à mente é a inundação na largura de banda DDoS (veja a Seção 1.6) contra servidores DNS. Por exemplo, um atacante pode tentar enviar para cada servidor DNS raiz uma inundação de pacotes, fazendo a maioria das consultas DNS legítimas nunca ser respondida. Um ataque DDoS em larga escala contra servidores DNS raiz aconteceu em 21 de outubro de 2002. Os atacantes aproveitaram um *botnet* para enviar centenas de mensagens *ping* para os 13 servidores DNS raiz. (As mensagens ICMP são discutidas no Capítulo 4. Por enquanto, basta saber que os pacotes ICMP são tipos especiais de datagramas IP.) Felizmente, esse ataque em larga escala causou um dano mínimo, tendo um pequeno ou nenhum impacto sobre a experiência dos usuários com a Internet. Os atacantes obtiveram êxito ao direcionar centenas de pacotes aos servidores raiz. Mas muitos dos servidores DNS raiz foram protegidos por filtros de pacotes, configurados para sempre bloquear todas as mensagens *ping* ICMP encaminhadas aos servidores raiz. Assim, esses servidores protegidos foram poupados e funcionaram normalmente. Além disso, a maioria dos servidores DNS locais oculta os endereços IP dos servidores de domínio de nível superior, permitindo que o processo de consulta ultrapasse com frequência os servidores DNS raiz.

Um ataque DDoS potencialmente mais eficaz contra o DNS seria enviar uma inundação de consultas DNS aos servidores de domínio de alto nível, por exemplo, para todos os que lidam com o domínio .com. Seria mais difícil filtrar as consultas DNS direcionadas aos servidores DNS; e os servidores de domínio de alto nível não são ultrapassados com tanta facilidade quanto os raiz. Mas a gravidade do ataque poderia ser em parte amenizada pelo *cache* nos servidores DNS locais.

O DNS poderia ser atacado de outras maneiras. Em um ataque de homem no meio, o atacante intercepta consultas do hospedeiro e retorna respostas falsas. No de envenenamento, o atacante envia respostas falsas a um servidor DNS, fazendo-o armazenar os registros falsos em sua *cache*. Ambos os ataques podem ser utilizados, por exemplo, para redirecionar um usuário da Web inocente ao site Web do atacante. Esses ataques, entretanto, são difíceis de implementar, uma vez que requerem a intercepção de pacotes ou o estrangulamento de servidores [Skoudis, 2006].

Outro ataque DNS importante não é um ataque ao serviço DNS por si mesmo, mas, em vez disso, se aproveitar da infraestrutura do DNS para lançar um ataque DDoS contra um hospedeiro-alvo (por exemplo, o servidor de mensagens de sua universidade). Nesse caso, o atacante envia consultas DNS para muitos servidores DNS autoritativos, com cada consulta tendo o endereço-fonte falsificado do hospedeiro-alvo. Os servidores DNS, então, enviam suas respostas diretamente para o hospedeiro-alvo. Se as consultas puderem ser realizadas de tal maneira que uma resposta seja muito maior (em bytes) do que uma consulta (denominada amplificação), então o atacante pode entupir o alvo sem ter que criar muito de seu próprio tráfego. Tais ataques de reflexão que exploram o DNS possuem um sucesso limitado até hoje [Mirkovic, 2005].

Em resumo, não houve um ataque que tenha interrompido o serviço DNS com sucesso. Houve ataques de reflexão bem-sucedidos; entretanto, eles podem ser (e estão sendo) abordados por uma configuração apropriada de servidores DNS.

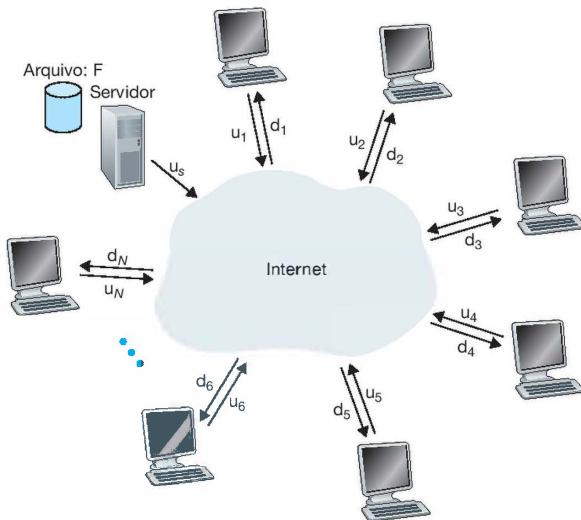
de sistema. Mais recentemente, foi acrescentada ao protocolo DNS uma opção UPDATE, que permite que dados sejam dinamicamente acrescentados no banco de dados ou apagados deles por meio de mensagens DNS. O [RFC 2136] e o [RFC 3007] especificam atualizações dinâmicas do DNS.)

Quando todas essas etapas estiverem concluídas, o público em geral poderá visitar seu site e enviar e-mails aos empregados de sua empresa. Vamos concluir nossa discussão do DNS verificando que essa afirmação é verdadeira, o

que também ajudará a solidificar aquilo que aprendemos sobre o DNS. Suponha que Alice, que está na Austrália, queira consultar a página `www.networkutopia.com`. Como discutimos, seu hospedeiro primeiro enviará uma consulta DNS a seu servidor de nomes local, que então contatará um servidor TLD do domínio com. (O servidor de nomes local também terá de contatar um servidor de nomes raiz caso não tenha em seu *cache* o endereço de um servidor TLD com.) Esse servidor TLD contém os registros de recursos Type NS e Type A já citados, porque a entidade registradora já os tinha inserido em todos os servidores TLD `com`. O servidor TLD com envia uma resposta ao servidor de nomes local de Alice, contendo os dois registros de recursos. Então, o servidor de nomes local envia uma consulta DNS a `212.212.212.1`, solicitando o registro Type A correspondente a `www.networkutopia.com`. Este registro oferece o endereço IP do servidor Web desejado, digamos, `212.212.71.4`, que o servidor local de nomes transmite para o hospedeiro de Alice. Agora, o navegador de Alice pode iniciar uma conexão TCP com o hospedeiro `212.212.71.4` e enviar uma requisição HTTP pela conexão. Ufa! Acontecem muito mais coisas do que percebemos quando navegamos na Web!

2.6 APLICAÇÕES P2P

Todas as aplicações descritas neste capítulo até agora — inclusive a Web, e-mail e DNS — empregam arquiteturas cliente-servidor com dependência significativa em servidores com infraestrutura que sempre permanecem ligados. Como consta na Seção 2.1.1, com uma arquitetura P2P, há dependência mínima (se houver) de servidores com infraestrutura que permanecem sempre ligados. Em vez disso, duplas de hospedeiros intermitentemente conectados, chamados pares, comunicam-se direto entre si. Os pares não são de propriedade de um provedor de serviços, mas sim de desktops e laptops controlados por usuários.

Nesta seção, examinaremos três diferentes aplicações que são particularmente bem apropriadas a projetos P2P. A primeira é a distribuição de arquivos, em que a aplicação distribui um arquivo a partir de uma única fonte para um grande número de pares. A distribuição de arquivos é um bom local para iniciar a investigação de P2P, visto que expõe com clareza a autoescalabilidade de arquiteturas P2P. Como exemplo específico para distribuição de arquivos, descreveremos o popular sistema BitTorrent. A segunda aplicação P2P que examinaremos é um banco de dados distribuído em uma grande comunidade de pares. Para essa aplicação, exploraremos o conceito de uma Distributed Hash Table (DHT).

2.6.1 Distribuição de arquivos P2P

Começaremos nossa investida em P2P considerando uma aplicação bastante natural, ou seja, a distribuição de um grande arquivo a partir de um único servidor a grande número de hospedeiros (chamados pares). O arquivo pode ser uma nova versão do sistema operacional Linux, um *patch* de software para um sistema operacional ou aplicação, um arquivo de música MP3 ou um arquivo de vídeo MPEG. Em uma distribuição de arquivo cliente-servidor, o servidor deve enviar uma cópia para cada par — colocando um enorme fardo sobre o servidor e consumindo grande quantidade de banda deste. Na distribuição de arquivos P2P, cada par pode redistribuir qualquer parte do arquivo recebido para outros pares, auxiliando, assim, o servidor no processo de distribuição. Hoje (em 2013), o protocolo de distribuição de arquivos P2P mais popular é o BitTorrent. Desenvolvido no início por Bram Cohen, há agora muitos diferentes clientes independentes de BitTorrent conformes ao protocolo do BitTorrent, assim como há diversos clientes de navegadores Web conformes ao protocolo HTTP. Nesta subseção, examinaremos primeiro a autoescalabilidade de arquiteturas P2P no contexto de distribuição de arquivos. Então, descreveremos o BitTorrent em certo nível de detalhes, destacando suas características mais importantes.

Escalabilidade de arquiteturas P2P

Para comparar arquiteturas cliente-servidor com arquiteturas P2P, e ilustrar a inerente autoescalabilidade de P2P, consideraremos um modelo quantitativo simples para a distribuição de um arquivo a um conjunto fixo

FIGURA 2.24 UM PROBLEMA ILUSTRATIVO DE DISTRIBUIÇÃO DE ARQUIVO

de pares para ambos os tipos de arquitetura. Conforme demonstrado na Figura 2.24, o servidor e os pares são conectados por enlaces de acesso da Internet. A taxa de *upload* do enlace de acesso do servidor é denotada por u_s, e a de *upload* do enlace de acesso do par *i* é denotada por u_i, e a taxa de *download* do enlace de acesso do par *i* é denotada por d_i. O tamanho do arquivo a ser distribuído (em bits) é indicado por *F* e o número de pares que querem obter uma cópia do arquivo, por *N*. O **tempo de distribuição** é o tempo necessário para que todos os *N* pares obtenham uma cópia do arquivo. Em nossa análise do tempo de distribuição a seguir, tanto para a arquitetura cliente-servidor como para a arquitetura P2P, fazemos a hipótese simplificada (e em geral precisa [Akella, 2003]) de que o núcleo da Internet tem largura de banda abundante, o que implica que todos os gargalos encontram-se no acesso à rede. Suponha também que o servidor e os clientes não participam de nenhuma outra aplicação de rede, para que toda sua largura de banda de acesso de *upload* e *download* possa ser totalmente dedicada à distribuição do arquivo.

Determinemos primeiro o tempo de distribuição para a arquitetura cliente-servidor, que indicaremos por D_{cs}. Na arquitetura cliente-servidor, nenhum dos pares auxilia na distribuição do arquivo. Fazemos as observações a seguir:

O servidor deve transmitir uma cópia do arquivo a cada um dos *N* pares. Assim, o servidor deve transmitir *NF* bits. Como a taxa de *upload* do servidor é de u_s, o tempo para distribuição do arquivo deve ser de pelo menos NF/u_s.

Suponha que d_{min} indique a taxa de *download* do par com menor taxa de *download*, ou seja, $d_{min} = \min\{d_1, d_p, ..., d_N\}$. O par com a menor taxa de *download* não pode obter todos os bits *F* do arquivo em menos de F/d_{min} segundos. Assim, o tempo de distribuição mínimo é de pelo menos F/d_{min}.

Reunindo essas observações, temos:

$$D_{cs} \geq \max\left\{\frac{NF}{u_s}, \frac{F}{d_{min}}\right\}.$$

Isso proporciona um limite inferior para o tempo mínimo de distribuição para a arquitetura cliente-servidor. Nos problemas do final do capítulo, você deverá demonstrar que o servidor pode programar suas transmis-

sões de forma que o limite inferior seja sempre alcançado. Portanto, consideraremos esse limite inferior fornecido antes como o tempo real de distribuição, ou seja,

$$D_{cs} = \max\left\{\frac{NF}{u_s}, \frac{F}{d_{min}}\right\} \tag{2.1}$$

Vemos, a partir da Equação 2.1, que para *N* grande o suficiente, o tempo de distribuição cliente-servidor é dado por NF/u_s. Assim, o tempo de distribuição aumenta linearmente com o número de pares *N*. Portanto, por exemplo, se o número de pares de uma semana para a outra for multiplicado por mil, de mil para um milhão, o tempo necessário para distribuir o arquivo para todos os pares aumentará mil vezes.

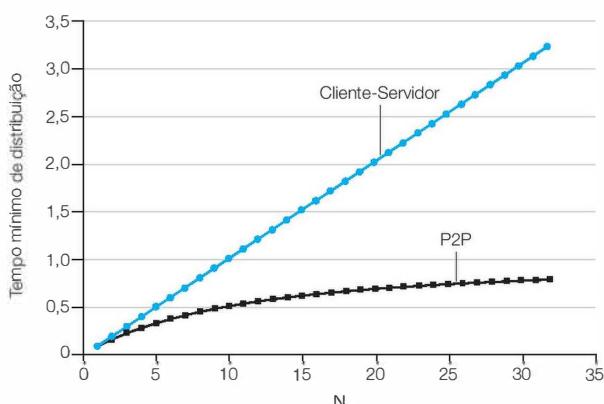
Passemos agora para uma análise semelhante para a arquitetura P2P, em que cada par pode auxiliar o servidor na distribuição do arquivo. Em particular, quando um par recebe alguns dados do arquivo, ele pode usar sua própria capacidade de *upload* para redistribuir os dados a outros pares. Calcular o tempo de distribuição para a arquitetura P2P é, de certa forma, mais complicado do que para a arquitetura cliente-servidor, visto que o tempo de distribuição depende de como cada par distribui parcelas do arquivo aos outros pares. Não obstante, uma simples expressão para o tempo mínimo de distribuição pode ser obtida [Kumar, 2006]. Para essa finalidade, faremos as observações a seguir:

- No início da distribuição, apenas o servidor tem o arquivo. Para levar esse arquivo à comunidade de pares, o servidor deve enviar cada bit do arquivo pelo menos uma vez para seu enlace de acesso. Assim, o tempo de distribuição mínimo é de pelo menos F/u_s. (Diferente do esquema cliente-servidor, um bit enviado uma vez pelo servidor pode não precisar ser enviado novamente, visto que os pares podem redistribuir entre si esse bit.)
- Assim como na arquitetura cliente-servidor, o par com a menor taxa de *download* não pode obter todos os bits *F* do arquivo em menos de F/d_{min} segundos. Assim, o tempo mínimo de distribuição é de pelo menos F/d_{min}.
- Por fim, observemos que a capacidade de *upload* total do sistema como um todo é igual à taxa de *upload* do servidor mais as taxas de *upload* de cada par, ou seja, $u_{total} = u_s + u_1 + \ldots + u_N$. O sistema deve entregar (fazer o *upload* de) *F* bits para cada um dos *N* pares, entregando assim um total de *NF* bits. Isso não pode ser feito em uma taxa mais rápida do que u_{total}. Assim, o tempo mínimo de distribuição é também de pelo menos $NF/(u_s + u_1 + \ldots + u_N)$.

Juntando as três observações, obtemos o tempo mínimo de distribuição para P2P, indicado por D_{P2P}.

$$D_{P2P} \geq \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\} \tag{2.2}$$

A Equação 2.2 fornece um limite inferior para o tempo mínimo de distribuição para a arquitetura P2P. Ocorre que, se imaginarmos que cada par pode redistribuir um bit assim que o recebe, há um esquema de redistribuição que, de fato, alcança esse limite inferior [Kumar, 2006] (Provaremos um caso especial desse resultado nos exercícios.) Na realidade, quando blocos do arquivo são redistribuídos, em vez de bits individuais, a Equação 2.2 serve como uma boa aproximação do tempo mínimo real de distribuição. Assim, peguemos o limite inferior fornecido pela Equação 2.2 como o tempo mínimo real de distribuição, que é:

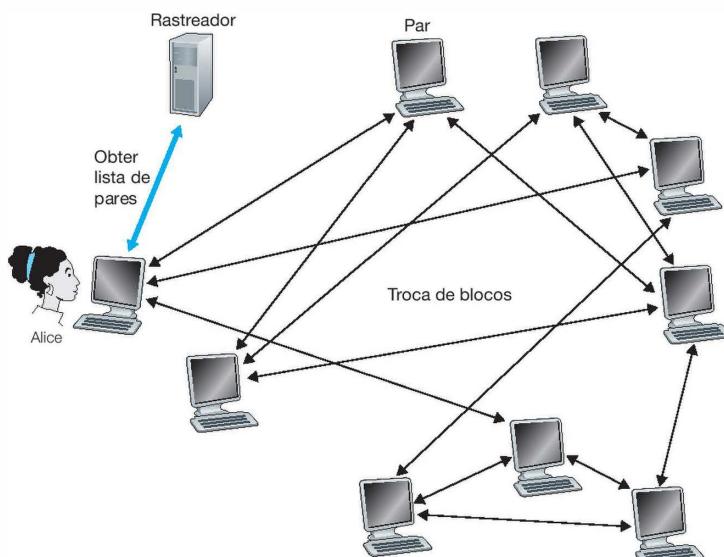
$$D_{\text{P2P}} = \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\} \tag{2.3}$$

A Figura 2.25 compara o tempo mínimo de distribuição para as arquiteturas cliente-servidor e P2P, pressupondo que todos os pares têm a mesma taxa de upload u. Na Figura 2.25, definimos que F/u = 1 hora, $u_s = 10u$ e $d_{\text{mín}} \geq u_s$. Assim, um par pode transmitir todo o arquivo em uma hora, sendo a taxa de transmissão do servidor 10 vezes a taxa de *upload* do par, e (para simplificar) as taxas de *download* de par são definidas grandes o suficiente de forma a não ter efeito. Vemos na Figura 2.25 que, para a arquitetura cliente-servidor, o tempo de distribuição aumenta linearmente e sem limite, conforme cresce o número de pares. No entanto, para a arquitetura P2P, o tempo mínimo de distribuição não é apenas sempre menor do que o tempo de distribuição da arquitetura cliente-servidor; é também de menos do que uma hora para *qualquer* número de pares N. Assim, aplicações com a arquitetura P2P podem ter autoescalabilidade. Tal escalabilidade é uma consequência direta de pares sendo redistribuidores, bem como consumidores de bits.

BitTorrent

O BitTorrent é um protocolo P2P popular para distribuição de arquivos [Chao, 2011]. No jargão do BitTorrent, a coleção de todos os pares que participam da distribuição de um determinado arquivo é chamada de *torrent*. Os pares em um *torrent* fazem o *download* de *blocos* de tamanho igual do arquivo entre si, com um tamanho típico de bloco de 256 KBytes. Quando um par entra em um *torrent*, ele não tem nenhum bloco. Com o tempo, ele acumula mais blocos. Enquanto ele faz o *download* de blocos, faz também *uploads* de blocos para outros pares. Uma vez que um par adquire todo o arquivo, ele pode (de forma egoísta) sair do *torrent* ou (de forma altruísta) permanecer e continuar fazendo o *upload* a outros pares. Além disso, qualquer par pode sair do *torrent* a qualquer momento com apenas um subconjunto de blocos, e depois voltar.

Observemos agora, mais atentamente, como opera o BitTorrent. Como é um protocolo e sistema complicado, descreveremos apenas seus mecanismos mais importantes, ignorando alguns detalhes; isso nos permitirá ver a floresta através das árvores. Cada *torrent* tem um nó de infraestrutura chamado *rastreador*. Quando um par chega em um *torrent*, ele se registra com o rastreador e periodicamente informa ao rastreador que ainda está lá.

FIGURA 2.25 TEMPO DE DISTRIBUIÇÃO PARA ARQUITETURAS P2P E CLIENTE-SERVIDOR

Dessa forma, o rastreador mantém um registro dos pares que participam do *torrent*. Um determinado *torrent* pode ter menos de dez ou mais de mil pares participando a qualquer momento.

Como demonstrado na Figura 2.26, quando um novo par, Alice, chega, o rastreador seleciona aleatoriamente um subconjunto de pares (para dados concretos, digamos que sejam 50) do conjunto de pares participantes, e envia os endereços IP desses 50 pares a Alice. Com a lista de pares, ela tenta estabelecer conexões TCP simultâneas com todos. Chamaremos todos os pares com quem Alice consiga estabelecer uma conexão TCP de "pares vizinhos". (Na Figura 2.26, Alice é representada com apenas três pares vizinhos. Normalmente, ela teria muito mais.) Com o tempo, alguns desses pares podem sair e outros pares (fora dos 50 iniciais) podem tentar estabelecer conexões TCP com Alice. Portanto, os pares vizinhos de um par podem flutuar com o tempo.

A qualquer momento, cada par terá um subconjunto de blocos do arquivo, com pares diferentes com subconjuntos diferentes. De tempos em tempos, Alice pedirá a cada um de seus pares vizinhos (nas conexões TCP) a lista de quais blocos eles têm. Caso Alice tenha L vizinhos diferentes, ela obterá L listas de blocos. Com essa informação, Alice emitirá solicitações (novamente, nas conexões TCP) de blocos que ela não tem.

Portanto, a qualquer momento, Alice terá um subconjunto de blocos e saberá quais blocos seus vizinhos têm. Com essa informação, ela terá duas decisões importantes a fazer. Primeiro, quais blocos deve solicitar de início a seus vizinhos, e segundo, a quais vizinhos deve enviar os blocos solicitados. Ao decidir quais blocos solicitar, Alice usa uma técnica chamada **rarest first** (o mais raro primeiro). A ideia é determinar, dentre os blocos que ela não tem, quais são os mais raros dentre seus vizinhos (ou seja, os blocos que têm o menor número de cópias repetidas) e então solicitar esses blocos mais raros primeiro. Dessa forma, os blocos mais raros são redistribuídos mais depressa, procurando (*grosso modo*) equalizar os números de cópias de cada bloco no torrent.

Para determinar a quais pedidos atender, o BitTorrent usa um algoritmo de troca inteligente. A ideia básica é Alice dar prioridade aos vizinhos que estejam fornecendo seus dados *com a maior taxa*. Especificamente, para cada vizinho, Alice mede de maneira contínua a taxa em que recebe bits e determina os quatro pares que lhe fornecem na taxa mais alta. Então, ela reciprocamente envia blocos a esses mesmos quatro pares. A cada 10 s, ela recalcula as taxas e talvez modifique o conjunto de quatro pares. No jargão do BitTorrent, esses quatro pares são chamados de **unchoked** (não sufocado). É importante informar que a cada 30 s ela também escolhe

FIGURA 2.26 DISTRIBUIÇÃO DE ARQUIVOS COM O BITTORRENT

um vizinho adicional ao acaso e envia blocos a ele. Chamaremos o vizinho escolhido de Bob. No jargão de BitTorrent, Bob é chamado de **otimisticamente não sufocado**. Como Alice envia dados a Bob, ela pode se tornar um dos quatro melhores transmissores para Bob, caso em que ele começaria a enviar dados para Alice. Caso a taxa em que Bob envie dados seja alta o suficiente, ele pode, em troca, tornar-se um dos quatro melhores transmissores para Alice. Em outras palavras, a cada 30 s, Alice escolherá ao acaso um novo parceiro de troca e a começará com ele. Caso os dois pares estejam satisfeitos com a troca, eles colocarão um ao outro nas suas listas de quatro melhores pares e continuarão a troca até que um dos pares encontre um parceiro melhor. O efeito é que pares capazes de fazer *uploads* em taxas compatíveis tendem a se encontrar. A seleção aleatória de vizinho também permite que novos pares obtenham blocos, de forma que possam ter algo para trocar. Todos os pares vizinhos, além desses cinco pares (quatro pares "top" e um em experiência) estão "sufocados", ou seja, não recebem nenhum bloco de Alice. O BitTorrent tem diversos mecanismos interessantes não discutidos aqui, incluindo pedaços (miniblocos), *pipelining* (tubulação), primeira seleção aleatória, modo *endgame* (fim de jogo) e *anti-snubbing* (antirrejeição) [Cohen 2003].

O mecanismo de incentivo para troca descrito costuma ser chamado de *tit-for-tat* (olho por olho) [Chen, 2003]. Demonstrou-se que esse esquema de incentivo pode ser burlado [Liogkas, 2006; Locher, 2006; Piatek, 2007]. Não obstante, o ecossistema do BitTorrent é muito bem-sucedido, com milhões de pares simultâneos compartilhando arquivos ativamente em centenas de milhares de *torrents*. Caso o BitTorrent tivesse sido projetado sem o *tit-for-tat* (ou uma variante), mas com o restante da mesma maneira, ele talvez nem existisse mais, visto que a maioria dos usuários são pessoas que apenas querem obter as coisas de graça [Saroiu, 2002].

Variantes interessantes do protocolo BitTorrent são propostas [Guo, 2005; Piatek, 2007]. Além disso, muitas das aplicações de transmissão em tempo real P2P, como PPLive e ppstream, foram inspiradas pelo BitTorrent [Hei, 2007].

2.6.2 Distributed Hash Tables (DHTs)

Nesta seção, vamos considerar como realizar um banco de dados simples em uma rede P2P. Começamos descrevendo uma versão centralizada desse banco de dados simples, que terá apenas pares (chave, valor). Por exemplo, as chaves podem ser números de seguridade social e os valores podem ser nomes humanos correspondentes; nesse caso, um exemplo de dupla chave-valor é (156-45-7081, Johnny Wu). Ou as duplas podem ser nomes de conteúdo (por exemplo, nomes de filmes, álbuns e software), e os valores podem ser endereços IP onde o conteúdo está armazenado; nesse caso, um exemplo de par chave-valor é (Led Zeppelin IV, 203.17.123.38). Pares consultam nossos bancos de dados fornecendo a chave: caso haja duplas (chave, valor) em seus bancos de dados que correspondam à chave, o banco de dados retorna as duplas correspondentes ao par solicitante. Portanto, por exemplo, se o banco de dados armazenar números de seguridade social e seus nomes humanos correspondentes, um par pode consultar um número de seguridade social e o banco de dados retornará o nome do humano que possui aquele número. Ou então, se o banco de dados armazenar os nomes de conteúdo e seus endereços IP correspondentes, podemos consultar um nome de conteúdo e o banco de dados retornará os endereços IP que armazenam aquele conteúdo.

Basear em tal banco de dados é simples com a arquitetura cliente-servidor que armazena todos os pares (chave, valor) em um servidor central. Assim, nesta seção, vamos considerar, em vez disso, como montar uma versão distribuída, P2P, desse banco de dados, que guardará os pares (chave, valor) por milhões. No sistema P2P, cada par só manterá um pequeno subconjunto da totalidade (chave, valor). Permitiremos que qualquer par consulte o banco de dados distribuído com uma chave em particular. O banco de dados distribuído, então, localizará os pares que possuem os pares (chave, valor) correspondentes e retornará os pares chave-valor ao consultante. Qualquer par também poderá inserir novos pares chave-valor no banco de dados. Esse banco de dados distribuído é considerado como uma **tabela *hash* distribuída (DHT — Distributed Hash Table)**.

Antes de descrever como podemos criar um DHT, primeiro vamos apresentar um exemplo específico de serviço DHT no contexto do compartilhamento de arquivos P2P. Neste caso, uma chave é o nome de conteúdo e

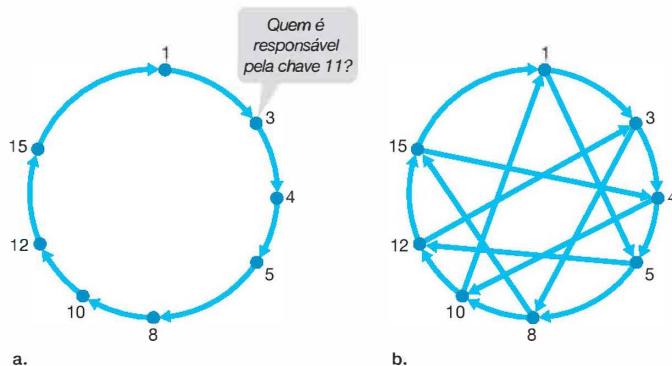
o valor é o endereço IP de um par que tem uma cópia do conteúdo. Assim, se Bob e Charlie tiverem cada um uma cópia da distribuição Linux mais recente, então o banco de dados DHT incluirá as seguintes duplas de chave-valor: (Linux, IP_{Bob}) e (Linux, $IP_{Charlie}$). Mais especificamente, como o banco de dados DHT é distribuído pelos pares, algum deles, digamos Dave, será responsável pela chave "Linux" e terá as duplas chave-valor correspondentes. Agora, suponha que Alice queira obter uma cópia do Linux. É claro, ela precisa saber primeiro quais pares têm uma cópia do Linux antes que possa começar a baixá-lo. Para essa finalidade, ela consulta o DHT com "Linux" como chave. O DHT, então, determina que Dave é responsável pela chave. O DHT entra em contato com Dave, obtém dele as duplas chave-valor (Linux, IP_{Bob}) e (Linux, $IP_{Charlie}$), e os passa a Alice. Ela pode, então, baixar a distribuição Linux mais recente a partir de IP_{Bob} ou de $IP_{Charlie}$.

Agora, vamos retornar ao problema de projetar um DHT para duplas gerais de chave-valor. Uma técnica ingênua para a criação de um DHT é espalhar ao acaso as duplas (chave, valor) por todos os pares e fazer cada um manter uma lista dos endereços IP de todos os pares. Nesse esquema, o par consultante envia sua consulta a todos os outros, e aqueles contendo as duplas (chave, valor) que combinam com a chave podem responder com suas duplas correspondentes. Essa técnica é totalmente não escalável, é claro, pois exigiria que cada par não apenas soubesse sobre todos os outros (talvez milhões deles!), mas, pior ainda, cada consulta deveria ser enviada a *todos* os pares.

Agora descreveremos uma abordagem elegante para o projeto de um DHT. Para isso, primeiro designaremos um identificador a cada par, em que cada identificador é um número inteiro na faixa $[0, 2^n - 1]$ de algum n fixo. Observe que cada identificador pode ser expresso por uma representação com n bits. Vamos também exigir que cada chave seja um número inteiro na mesma faixa. O leitor atento pode ter observado que as chaves de exemplo descritas (números de seguridade social e nomes de conteúdo) não são números inteiros. Para criar números inteiros a partir delas, precisaremos usar uma função *hash* que mapeie cada chave (por exemplo, número de seguridade social) em um número inteiro na faixa $[0, 2^n - 1]$. Uma função de *hash* é uma função de muitos-para-um para a qual duas entradas diferentes podem ter a mesma saída (mesmo número inteiro), mas a probabilidade de terem a mesma saída é extremamente pequena. (Leitores não familiarizados com funções de *hash* podem querer consultar o Capítulo 8, que as discute em detalhes.) A função de *hash* é considerada publicamente disponível a todos os pares no sistema. Daqui em diante, quando nos referirmos à "chave", estaremos nos referindo ao *hash* da chave original. Portanto, por exemplo, caso a chave original seja "Led Zeppelin IV", a chave usada no DHT será o número inteiro que corresponda ao *hash* de "Led Zeppelin IV". Como você já deve ter percebido, é por isso que "Hash" é usado no termo "Distributed Hash Table".

Consideraremos agora o problema de armazenar as duplas (chave, valor) no DHT. A questão central aqui é definir uma regra para designar chaves a pares. Considerando que cada par tenha um identificador de número inteiro e cada chave também seja um número inteiro na mesma faixa, uma abordagem natural é designar cada dupla (chave, valor) ao par cujo identificador está *mais próximo* da chave. Para executar esse esquema, precisaremos definir o que significa "mais próximo", o que admite muitas convenções. Por conveniência, definiremos que o par mais próximo é o *sucessor imediato da chave*. Para entender melhor, observaremos um exemplo. Considere que $n = 4$, portanto, todos os identificadores de par e chave estarão na faixa de [0, 15]. Suponha ainda que haja oito pares no sistema com identificadores 1, 3, 4, 5, 8, 10, 12 e 15. Por fim, imagine que queiramos armazenar a dupla chave-valor (11, Johnny Wu) em um dos oito pares. Mas em qual? Usando nossa convenção de mais próximo, como o par 12 é o sucessor imediato da chave 11, armazenaremos, portanto, a dupla (11, Johnny Wu) no par 12. [Para concluir nossa definição de mais próximo, caso a chave seja idêntica a um dos identificadores do par, armazenaremos a dupla (chave-valor) em um par correspondente; e caso seja maior do que todos os identificadores de par, usaremos uma convenção módulo-2^n, que armazena a dupla (chave-valor) no par com o menor identificador.]

Suponha agora que um par, Alice, queira inserir uma dupla (chave-valor) no DHT. Na concepção, é um processo objetivo: ela primeiro determina o par cujo identificador é o mais próximo da chave; então envia uma mensagem a esse par, instruindo-o a armazenar a dupla (chave, valor). Mas como Alice determina o par mais próximo da chave? Se ela rastreasse todos os pares no sistema (IDs de par e endereços IP correspondentes), poderia determinar localmente o par mais próximo. Mas essa abordagem requer que *cada* par rastreie *todos* os outros pares no DHT — o que é completamente impraticável para um sistema de grande escala com milhões de pares.

DHT circular

Para tratar desse problema de escala, consideraremos agora organizar os pares em um círculo. Nessa disposição circular, cada par rastreia apenas seu sucessor e predecessor imediatos (módulo 2^n). Um exemplo desse círculo é exibido na Figura 2.27(a). Nesse exemplo, *n* é novamente 4 e há os mesmos oito pares do exemplo anterior. Cada par está ciente apenas de seu sucessor e predecessor imediatos; por exemplo, o par 5 sabe o endereço IP e o identificador do par 8, mas talvez não saiba nada sobre quaisquer outros pares no DHT. Essa disposição circular dos pares é um caso especial de uma **rede sobreposta**. Nesse tipo de rede, os pares formam uma rede lógica abstrata que reside acima da de computadores "inferior" que consiste de enlaces físicos, roteadores e hospedeiros. Os enlaces em uma rede sobreposta não são físicos, mas enlaces virtuais entre duplas de pares. Na rede de sobreposição da Figura 2.27(a), há oito pares e oito enlaces sobrepostos; na sobreposição da Figura 2.27(b) há oito pares e 16 enlaces sobrepostos. Um único enlace de sobreposição em geral usa muitas ligações físicas e roteadores físicos na rede inferior.

Usando a rede de sobreposição circular da Figura 2.27(a), suponha agora que o par 3 deseje determinar qual par no DHT é responsável pela chave 11. Usando a rede sobreposta circular, o par de origem (par 3) cria uma mensagem que pergunta "Quem é responsável pela chave 11?" e a envia no sentido horário ao redor do círculo. Sempre que um par recebe essa mensagem, como conhece o identificador de seu sucessor e predecessor, pode determinar se é responsável pela (ou seja, mais próximo da) chave em questão. Caso um par não seja responsável pela chave, ele apenas envia a mensagem a seu sucessor. Portanto, por exemplo, quando o par 4 recebe a mensagem perguntando sobre a chave 11, ele determina que não é responsável pela chave (porque seu sucessor está mais perto dela), portanto, ele passa a mensagem a seu sucessor, ou seja, o par 5. O processo continua até que a mensagem chegue ao par 12, que determina que é o mais próximo da chave 11. A essa altura, o par 12 pode enviar uma mensagem de volta à origem, o par 3, indicando que é responsável pela chave 11.

O DHT circular oferece uma solução bastante elegante para reduzir a quantidade de informação sobreposta que cada par deve gerenciar. Em particular, cada par está ciente apenas de dois outros, seu sucessor e seu predecessor imediato. Porém, essa solução ainda introduz um novo problema. Embora cada par esteja ciente de dois pares vizinhos, para encontrar o nó responsável por uma chave (no pior das hipóteses), todos os *N* nós no DHT deverão encaminhar uma mensagem pelo círculo; *N*/2 mensagens são enviadas em média.

Assim, no projeto de um DHT, há uma escolha entre o número de vizinhos que cada par tem de rastrear e o número de mensagens que o DHT precisa enviar para resolver uma única solicitação. Por um lado, se cada par rastrear todos os outros (sobreposições de malha), apenas uma mensagem será enviada por solicitação, mas cada par deverá rastrear *N* pares. Por outro lado, com um DHT circular, cada par está ciente apenas de dois, mas *N/2* mensagens são enviadas em média por solicitação. Felizmente, podemos refinar nossos proje-

FIGURA 2.27 **(A) UM DHT CIRCULAR. O PAR 3 QUER DETERMINAR QUEM É RESPONSÁVEL PELA CHAVE 11. (B) UM DHT CIRCULAR COM ATALHOS**

tos de DHTs de forma que o número de vizinhos por par, bem como o número de mensagens por solicitação seja mantido em um tamanho aceitável. Um desses refinamentos é usar a rede sobreposta circular como fundação, mas adicionar "atalhos" de forma que cada par não apenas rastreie seu sucessor imediato, mas também um número relativamente pequeno de pares espalhados pelo círculo. Um exemplo de DHT circular com alguns atalhos é mostrado na Figura 2.27(b). Atalhos são usados para expedir o roteamento das mensagens de solicitação. Especificamente, quando um par recebe uma mensagem que solicita uma chave, ele encaminha a mensagem ao vizinho (sucessor ou um dos vizinhos via atalho) que está mais perto da chave. Assim, na Figura 2.27(b), quando o par 4 recebe a mensagem solicitando a chave 11, ele determina que o par mais próximo (entre seus vizinhos) é seu vizinho no atalho 10 e então envia a mensagem diretamente ao par 10. É claro que atalhos podem reduzir de modo significativo o número de mensagens usado para processar uma solicitação.

A próxima questão natural é "Quantos vizinhos no atalho cada par deve ter, e quais pares devem ser esses vizinhos de atalho?". A pergunta recebeu atenção significativa da comunidade de pesquisa [Balakrishnan, 2003; Androutsellis-Theotokis, 2004]. De forma importante, demonstrou-se que o DHT pode ser projetado para que tanto o número de vizinhos como o de mensagens por solicitação seja da ordem de O(log N), em que N é o número de pares. Esses projetos obtêm um compromisso satisfatório entre as soluções extremas de se usar topologias de sobreposição circular e de malha.

Peer Churn

Em sistemas P2P, um par pode vir ou ir sem aviso. Assim, no projeto de um DHT, devemos nos preocupar em manter a sobreposição de DHT na presença desse *peer churn*. Para termos uma compreensão abrangente de como isso pode ser realizado, consideraremos mais uma vez o DHT circular da Figura 2.27(a). Para tratar do *peer churn*, exigiremos que cada par rastreie (ou seja, saiba o endereço IP de) seu primeiro e segundo sucessores; por exemplo, o par 4 agora rastreia tanto o 5 como o 8. Exigiremos também que cada par verifique de tempos em tempos se seus dois sucessores estão vivos (por exemplo, enviando mensagens de *ping* e pedindo respostas). Consideraremos agora como o DHT é mantido quando um par sai bruscamente. Por exemplo, suponha que o par 5 da Figura 2.27(a) saia de modo abrupto. Nesse caso, os dois pares precedentes ao que saiu (4 e 3) saberão que o par saiu, pois não responde mais às mensagens de *ping*. Os pares 4 e 3 precisam, portanto, atualizar as informações do estado de seu sucessor. Consideraremos agora como o par 4 atualiza seu estado:

1. O par 4 substitui seu primeiro sucessor (par 5) por seu segundo sucessor (par 8).
2. O par 4, então, pergunta a seu novo primeiro sucessor (par 8) o identificador e o endereço IP de seu sucessor imediato (par 10). O par 4, então, torna o par 10 seu segundo sucessor.

Nos problemas, você deverá determinar como o par 3 atualiza suas informações de determinação de roteamento de sobreposição.

Tendo abordado de modo sucinto o que deve ser feito quando um par sai, consideraremos agora o que acontece quando um par quer entrar no DHT. Digamos que um par com identificador 13 quer entrar, e quando entra, sabe apenas da existência do par 1 no DHT. O par 13 primeiro envia ao par 1 uma mensagem, perguntando "quem serão o predecessor e o sucessor do par 13?". Essa mensagem é encaminhada pelo DHT até alcançar o par 12, que percebe que será o predecessor do par 13, e que seu sucessor, o par 15, será o sucessor do par 13. Em seguida, o par 12 envia as informações de sucessor e predecessor ao par 13. Este, então, pode entrar no DHT, tornando o par 15 seu sucessor e notificando ao par 12 que deve mudar seu sucessor imediato para 13.

DHTs têm amplo uso na prática. Por exemplo, o BitTorrent usa o DHT Kademlia para criar um rastreador distribuído. No BitTorrent, a chave é o identificador do *torrent* e o valor é o endereço IP dos pares que atualmente participam dos *torrents* [Falkner, 2007; Neglia, 2007]. Dessa forma, solicitando ao DHT um identificador de *torrent*, um par de BitTorrent recém-chegado pode determinar o par responsável pelo identificador (ou seja, por determinar os pares no *torrent*). Após ter encontrado esse par, o recém-chegado pode solicitar dele uma lista de outros pares no *torrent*.

2.7 Programação de *sockets*: criando aplicações de rede

Agora que já examinamos várias importantes aplicações de rede, vamos explorar como são escritos programas de aplicação de rede. Lembre-se de que na Seção 2.1 dissemos que muitas aplicações de rede consistem em um par de programas — um programa cliente e um programa servidor — que residem em dois sistemas finais diferentes. Quando são executados, criam-se um processo cliente e um processo servidor, que se comunicam entre si lendo de seus *sockets* e escrevendo através deles. Ao criar uma aplicação de rede, a tarefa principal do programador é escrever o código tanto para o programa cliente como para o programa servidor.

Há dois tipos de aplicações de rede. Um deles é uma execução cuja operação é especificada em um padrão de protocolo, por exemplo, em um RFC ou algum outro documento de padrões; essa aplicação às vezes é denominada "aberta", pois as regras especificando sua operação são conhecidas de todos. Para essa implementação, os programas, cliente e servidor, devem obedecer às regras ditadas pelo RFC. Por exemplo, o programa cliente poderia ser uma execução do lado do cliente do protocolo FTP descrito na Seção 2.3 e definido explicitamente no RFC 959 e o programa servidor, uma implementação do protocolo de servidor FTP também descrito de modo explícito no RFC 959. Se um programador escrever um código para o programa cliente e outro programador independente escrever um código para o programa servidor e ambos seguirem com atenção as regras do RFC, então os dois programas poderão interagir. De fato, muitas das aplicações de rede de hoje envolvem comunicação entre programas cliente e servidor que foram criados por programadores diferentes — por exemplo, um navegador Firefox que se comunica com um servidor Web Apache, ou um cliente BitTorrent que se comunica com um rastreador BitTorrent.

O outro tipo de aplicação de rede é uma aplicação de rede proprietária. Nesse caso, os programas cliente e servidor empregam um protocolo de camada de aplicação que *não* foi publicado abertamente em um RFC ou em outro lugar. Um único programador (ou equipe de desenvolvimento) cria ambos os programas cliente e servidor, e tem completo controle sobre o que entra no código. Mas, como o código não implementa um protocolo de domínio público, outros programadores independentes não poderão desenvolver código que interage com a aplicação.

Nesta seção e na próxima, examinaremos as questões fundamentais do desenvolvimento de uma aplicação cliente-servidor, e "sujaremos nossas mãos" examinando o código que executa uma aplicação cliente-servidor muito simples. Durante a fase de desenvolvimento, uma das primeiras decisões que o programador deve tomar é se a aplicação rodará em TCP ou UDP. Lembre-se de que o TCP é orientado para conexão e provê um canal confiável de cadeia de bytes, pelo qual fluem dados entre dois sistemas finais. O UDP não é orientado para conexão e envia pacotes de dados independentes de um sistema final ao outro, sem nenhuma garantia de entrega. Lembre-se também que, quando um programa, cliente ou servidor, executa um protocolo definido em um RFC, deve usar o número de porta conhecido associado com o protocolo; por outro lado, ao desenvolver uma aplicação proprietária, o programador deve ter o cuidado de evitar esses números de porta conhecidos. (Números de portas foram discutidos brevemente na Seção 2.1. Serão examinados mais detalhadamente no Capítulo 3.)

Apresentamos a programação de *sockets* UDP e TCP por meio de aplicações UDP e TCP simples em Python. Poderíamos escrevê-las em linguagem Java, C ou C++, mas optamos por Python por diversas razões, principalmente porque Python expõe com clareza os conceitos principais de *sockets*. Com Python, há menos linhas de codificação e cada uma delas pode ser explicada a programadores iniciantes sem muita dificuldade. Mas não precisa ficar assustado se não estiver familiarizado com a linguagem. Você conseguirá acompanhar o código com facilidade se tiver experiência de programação em Java, C ou C++.

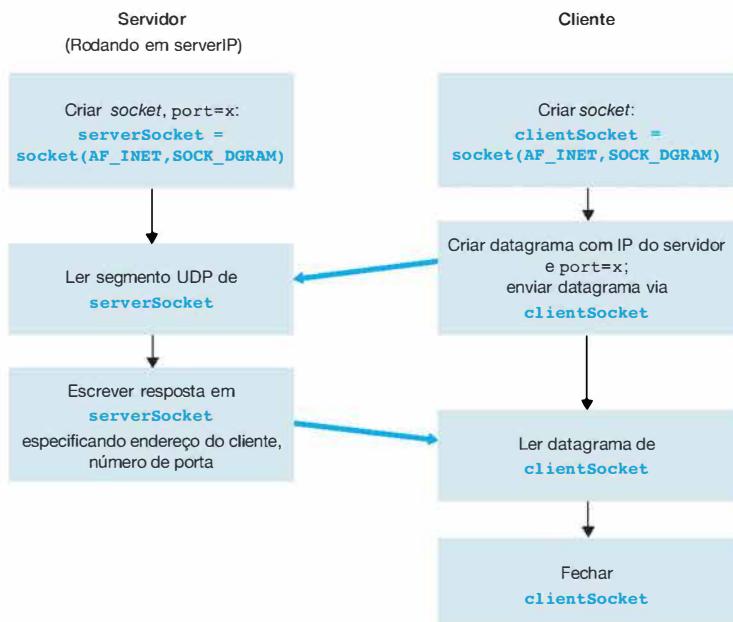
Se estiver interessado em programação cliente-servidor em linguagem Java, veja o site de apoio que acompanha este livro; lá você poderá achar todos os exemplos desta seção (e laboratórios associados) em Java. Para os interessados em programação cliente-servidor em C, há várias boas referências à disposição [Donahoo, 2001; Stevens, 1997; Frost, 1994; Kurose, 1996]; nossos exemplos em Python a seguir possuem um estilo semelhante a C.

2.7.1 Programação de *sockets* com UDP

Nesta subseção, vamos escrever programas cliente-servidor simples que usam UDP; na próxima, escreveremos programas semelhantes que usam TCP.

Comentamos na Seção 2.1 que processos que rodam em máquinas diferentes se comunicam entre si enviando mensagens para *sockets*. Dissemos que cada processo é semelhante a uma casa e que o *socket* do processo é semelhante a uma porta. A aplicação reside em um lado da porta na casa; o protocolo da camada de transporte reside no outro lado da porta, no mundo exterior. O programador da aplicação controla tudo que está no lado da camada de aplicação da porta; contudo, tem pouco controle do lado da camada de transporte.

Agora, vejamos mais de perto a interação entre dois processos que se comunicam, que utilizam *sockets* UDP. Antes que o processo emissor consiga empurrar um pacote de dados pela porta do *socket*, ao usar UDP, ele deve primeiro incluir um endereço de destino no pacote. Depois que o pacote passa pelo *socket* do emissor, a Internet usará esse endereço de destino para rotear o pacote pela Internet até o *socket* no processo receptor. Quando o pacote chega no *socket* receptor, o processo receptor apanha o pacote através do *socket* e depois inspeciona o conteúdo do pacote e toma a ação apropriada.

Assim, você pode agora querer saber: o que há no endereço de destino que é acrescentado ao pacote? Como é de se esperar, o endereço IP do hospedeiro de destino faz parte do endereço de destino. Ao incluir o endereço IP do destino no pacote, os roteadores na Internet poderão rotear o pacote pela Internet até o hospedeiro de destino. Mas, como o hospedeiro pode estar rodando muitos processos de aplicação de rede, cada um com um ou mais *sockets*, também é preciso identificar o *socket* em particular no hospedeiro de destino. Quando um *socket* é criado, um identificador, chamado **número de porta**, é designado para ele. Assim, como é de se esperar, o endereço de destino do pacote também inclui o número de porta do *socket*. Resumindo, o processo emissor inclui no pacote um endereço de destino que consiste no endereço IP do hospedeiro de destino e o número de porta do *socket* de destino. Além do mais, como veremos em breve, o endereço de origem do emissor — consistindo no endereço IP do hospedeiro de destino e o número de porta do *socket* de origem — também é acrescentado ao pacote. Porém, a inclusão do endereço de origem ao pacote normalmente *não* é feita pelo código da aplicação UDP; em vez disso, ela é feita automaticamente pelo sistema operacional.

Usaremos a aplicação cliente-servidor simples a seguir para demonstrar a programação de *socket* para UDP e TCP:

1. Um cliente lê uma linha de caracteres (dados) do teclado e a envia para o servidor.
2. O servidor recebe os dados e converte os caracteres para maiúsculas.
3. O servidor envia os dados modificados ao cliente.
4. O cliente recebe os dados modificados e apresenta a linha em sua tela.

A Figura 2.28 destaca a principal atividade relacionada ao *socket* realizada pelo cliente e pelo servidor, que se comunicam por meio de um serviço de transporte.

Agora vamos pôr as mãos na massa e dar uma olhada no par de programas cliente-servidor para uma implementação UDP dessa aplicação de exemplo. Também oferecemos uma análise detalhada, linha a linha, após cada programa. Começamos com um cliente UDP, que enviará uma mensagem simples, em nível de aplicação, ao servidor. Para que o servidor possa receber e responder à mensagem do cliente, ele precisa estar pronto e rodando — ou seja, precisa estar rodando como um processo antes que o cliente envie sua mensagem.

O programa cliente é denominado UDPClient.py e o programa servidor é denominado UDPServer.py. Para enfatizar os principais pontos, oferecemos intencionalmente um código que seja mínimo. Um "código bom" certamente teria muito mais linhas auxiliares, particularmente para tratar de casos de erro. Para esta aplicação, escolhemos arbitrariamente 12000 para o número de porta do servidor.

FIGURA 2.28 A APLICAÇÃO CLIENTE-SERVIDOR USANDO UDP

UDPClient.py

Aqui está o código para o lado cliente da aplicação:

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.close()
```

Agora, vamos examinar as linhas de código em UDPClient.py.

```
from socket import *
```

O módulo *socket* forma a base de todas as comunicações de rede em Python. Incluindo esta linha, poderemos criar *sockets* dentro do nosso programa.

```
serverName = 'hostname'
serverPort = 12000
```

A primeira linha define a cadeia `serverName` como "hostname". Aqui, oferecemos uma cadeia contendo ou o endereço IP do servidor (por exemplo, "128.138.32.126") ou o nome de hospedeiro do servidor (por exemplo, "cis.poly.edu"). Se usarmos o nome do hospedeiro, então uma pesquisa DNS será automaticamente realizada para obter o endereço IP. A segunda linha define a variável inteira `serverPort` como 12000.

```
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
```

Esta linha cria o *socket* do cliente, denominado `clientSocket`. O primeiro parâmetro indica a família do endereço; em particular, `AF_INET` indica que a rede subjacente está usando IPv4. (Não se preocupe com isso agora — vamos discutir sobre o IPv4 no Capítulo 4.) O segundo parâmetro indica que o *socket* é do tipo `SOCK_DGRAM`, o que significa que é um *socket* UDP (em vez de um *socket* TCP). Observe que não estamos especificando o número de porta do *socket* cliente quando o criamos; em vez disso, deixamos que o sistema operacional o faça por nós. Agora que a porta do processo cliente já foi criada, queremos criar uma mensagem para enviar pela porta.

```
message = raw_input('Input lowercase sentence:')
```

`raw_input()` é uma função interna da linguagem Python. Quando esse comando é executado, o usuário no cliente recebe o texto "`Input data:`" (digite dados). Então, o usuário usa seu teclado para digitar uma linha, que é colocada na variável `message`. Agora que temos um *socket* e uma mensagem, queremos enviar a mensagem pelo *socket* ao hospedeiro de destino.

```
clientSocket.sendto(message,(serverName, serverPort))
```

Nesta linha, o método `sendto()` acrescenta o endereço de destino (`serverName, serverPort`) à mensagem e envia o pacote resultante pelo *socket* do processo, `clientSocket`. (Como já dissemos, o endereço de origem também é conectado ao pacote, embora isso seja feito automaticamente, e não pelo código.) O envio de uma mensagem do cliente ao servidor por meio de um *socket* UDP é simples assim! Depois de enviar o pacote, o cliente espera receber dados do servidor.

```
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
```

Com esta linha, quando um pacote chega da Internet no *socket* do cliente, os dados são colocados na variável `modifiedMessage` e o endereço de origem do pacote é colocado na variável `serverAddress`. A variável `serverAddress` contém tanto o endereço IP do servidor quanto o número de porta do servidor. O programa UDPClient não precisa realmente dessa informação de endereço do servidor, pois já sabe o endereço do servidor desde o início; mas esta linha de Python oferece o endereço do servidor, apesar disso. O método recvfrom também toma o tamanho do buffer, 2048, como entrada. (Esse tamanho de buffer funciona para quase todos os fins.)

```
print modifiedMessage
```

Esta linha imprime modifiedMessage na tela do usuário. Essa deverá ser a linha original que o usuário digitou, mas agora em letras maiúsculas.

```
clientSocket.close()
```

Esta linha fecha o *socket*. O processo, então, é concluído.

UDPServer.py

Vamos agora dar uma olhada no lado servidor da aplicação:

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:

        message, clientAddress = serverSocket.recvfrom(2048)

        modifiedMessage = message.upper()
        serverSocket.sendto(modifiedMessage, clientAddress)
```

Observe que o início de UDPServer é semelhante a UDPClient. Ele também importa o módulo *socket*, também define a variável inteira `serverPort` como 12000 e também cria um *socket* do tipo `SOCK_DGRAM` (um *socket* UDP). A primeira linha de código que é significativamente diferente de UDPClient é:

```
serverSocket.bind(('', serverPort))
```

Esta linha vincula (ou seja, designa) o número de porta 12000 ao *socket* do servidor. Assim, em UDPServer, o código (escrito pelo programador de aplicação) está designando um número de porta ao *socket*. Dessa forma, quando alguém enviar um pacote à porta 12000 no endereço IP do servidor, ele será direcionado a este *socket*. UDPServer, então, entra em um laço *while*; o laço *while* permitirá que UDPServer receba e processe pacotes dos clientes indefinidamente. No laço *while*, UDPServer espera um pacote chegar.

```
message, clientAddress = serverSocket.recvfrom(2048)
```

Esta linha de código é semelhante à que vimos em UDPClient. Quando um pacote chega no *socket* do servidor, os dados são colocados na variável message e o endereço de origem é colocado na variável `clientAddress`. A variável clientAddress contém o endereço IP e o número de porta do cliente. Aqui, UDPServer *usará* essa informação de endereço, pois oferece um endereço de retorno, semelhante ao do remetente no serviço postal comum. Com essa informação, o servidor agora sabe para onde deve direcionar sua resposta.

```
modifiedMessage = message.upper()
```

Esta linha é o núcleo da nossa aplicação simples. Ela apanha a linha enviada pelo cliente e usa o método upper() para passá-la para letras maiúsculas.

```
serverSocket.sendto(modifiedMessage, clientAddress)
```

Esta última linha anexa o endereço do cliente (endereço IP e número de porta) à mensagem em letras maiúsculas, enviando o pacote resultante ao *socket* do servidor. (Como já dissemos, o endereço do servidor também é anexado ao pacote, embora isso seja feito automaticamente, e não pelo código.) A Internet, então, entregará o pacote a esse endereço do cliente. Depois que o servidor envia o pacote, ele permanece no laço *while*, esperando até que outro pacote UDP chegue (de qualquer cliente rodando em qualquer hospedeiro).

Para testar o par de programas, você instala e compila UDPClient.py em um hospedeiro e UDPServer.py em outro. Não se esqueça de incluir o nome de hospedeiro ou endereço IP do servidor em UDPClient.py. Em seguida, executa UDPServer.py, o programa servidor compilado, no hospedeiro servidor. Isso cria um processo no servidor que fica ocioso até que seja chamado por algum cliente. Depois você executa UDPClient.py, o programa cliente compilado, no cliente. Isso cria um processo no cliente. Por fim, para usar a aplicação no cliente, você digita uma sentença seguida por um Enter.

Para desenvolver sua própria aplicação cliente-servidor UDP, você pode começar modificando um pouco os programas cliente e servidor. Por exemplo, em vez de converter todas as letras para maiúsculas, o servidor poderia contar o número de vezes que a letra *s* aparece e retornar esse número. Ou então o cliente pode ser modificado para que, depois de receber uma sentença em maiúsculas, o usuário possa continuar a enviar mais sentenças ao servidor.

2.7.2 Programação de *sockets* com TCP

Diferente do UDP, o TCP é um protocolo orientado a conexão. Isso significa que, antes que cliente e servidor possam começar a enviar dados um para o outro, eles precisam primeiro se apresentar e estabelecer uma conexão TCP. Uma ponta dessa conexão está ligada ao *socket* cliente e a outra está ligada a um *socket* servidor. Ao criar a conexão TCP, associamos a ela o endereço de *socket* (endereço IP e número de porta) do cliente e do servidor. Com a conexão estabelecida, quando um lado quer enviar dados para o outro, basta deixá-los na conexão TCP por meio de seu *socket*. Isso é diferente do UDP, para o qual o servidor precisa anexar um endereço de destino ao pacote, antes de deixá-lo no *socket*.

Agora, vamos examinar mais de perto a interação dos programas cliente e servidor em TCP. O cliente tem a tarefa de iniciar contato com o servidor. Para que o servidor possa reagir ao contato inicial do cliente, ele tem de estar pronto, o que implica duas coisas. Primeiro, como acontece no UDP, o programa servidor TCP precisa estar rodando como um processo antes de o cliente tentar iniciar contato. Segundo, o programa servidor tem de ter alguma porta especial — mais precisamente, um *socket* especial — que acolha algum contato inicial de um processo cliente que esteja rodando em um hospedeiro qualquer. Recorrendo à analogia casa/porta para processo/*socket*, às vezes nos referiremos ao contato inicial do cliente como "bater à porta".

Com o processo servidor em execução, o processo cliente pode iniciar uma conexão TCP com o servidor, o que é feito no programa cliente pela criação de um *socket* TCP. Quando cria seu *socket* TCP, o cliente especifica o endereço do *socket* receptivo do servidor, a saber, o endereço IP do hospedeiro servidor e o número de porta do *socket*. Após a criação de seu *socket*, o cliente inicia uma apresentação de três vias e estabelece uma conexão TCP com o servidor. Essa apresentação, que ocorre dentro da camada de transporte, é toda invisível para os programas cliente e servidor.

Durante a apresentação de três vias, o processo cliente bate na porta de entrada do processo servidor. Quando o servidor "ouve" a batida, cria uma nova porta (mais precisamente, um *novo socket*) dedicada àquele cliente. No exemplo a seguir, a porta de entrada é um objeto *socket* do TCP que denominamos serverSocket; o *socket* recém-criado, dedicado ao cliente que faz a conexão, é denominado connectionSocket. Os estudantes que encontram *sockets* TCP pela primeira vez às vezes confundem o *socket* de entrada (que é o ponto de contato inicial para todos os clientes que querem se comunicar com o servidor) com cada *socket* de conexões no lado do servidor, que é criado em seguida para a comunicação com cada cliente.

Do ponto de vista da aplicação, o *socket* do cliente e o de conexão do servidor estão conectados diretamente, como se houvesse uma tubulação entre eles. Como vemos na Figura 2.29, o processo cliente pode enviar bytes para seu *socket* de modo arbitrário; o TCP garante que o processo servidor receberá (pelo *socket* de conexão) cada byte na ordem em que foram enviados. Assim, o TCP provê um serviço confiável entre os processos cliente e servidor. Além disso, assim como pessoas podem entrar e sair pela mesma porta, o processo cliente não somente envia bytes a seu *socket*, mas também os recebe dele; de modo semelhante, o processo servidor não só recebe bytes de seu *socket* de conexão, mas também os envia por ele.

Usamos a mesma aplicação cliente-servidor simples para demonstrar programação de *sockets* para TCP: o cliente envia uma linha de dados ao servidor, este converte a linha para letras maiúsculas e a envia de volta ao cliente. A Figura 2.30 destaca a principal atividade relacionada a *socket* do cliente e servidor que se comunicam pelo serviço de transporte TCP.

FIGURA 2.29 O PROCESSO TCPSERVER TEM DOIS *SOCKETS*

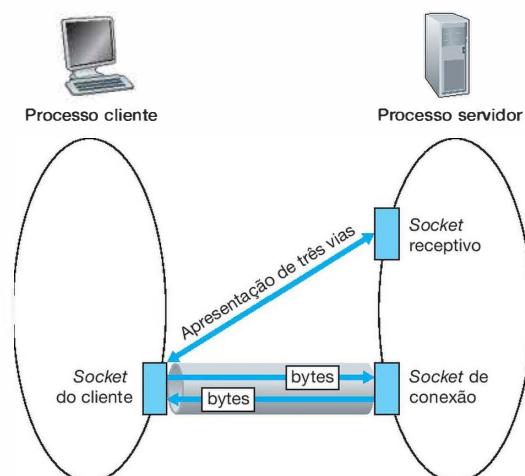
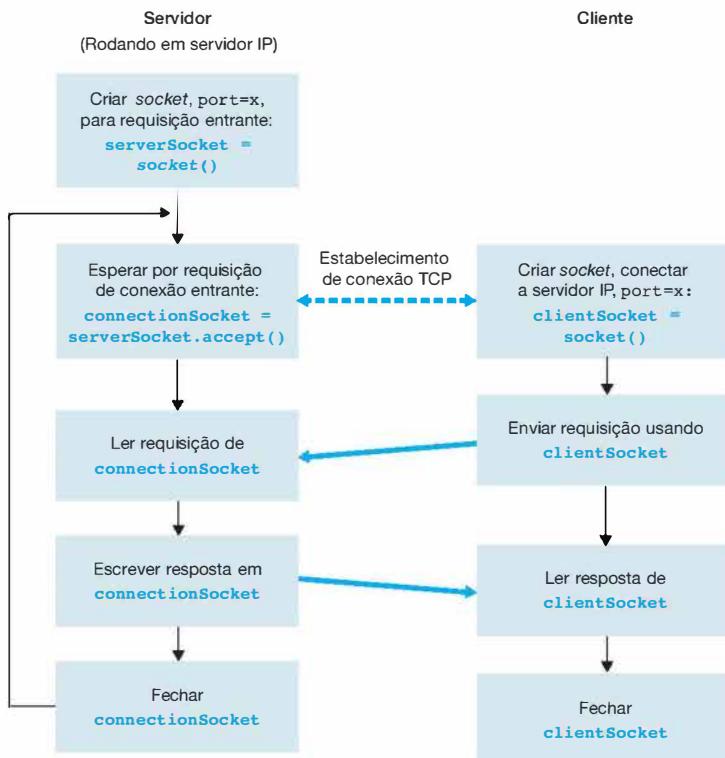



FIGURA 2.30 A APLICAÇÃO CLIENTE-SERVIDOR USANDO TCP

TCPClient.py

Eis o código para o lado cliente da aplicação:

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

Vamos agora examinar as várias linhas do código, que difere ligeiramente da implementação UDP. A primeira linha é a criação do *socket* do cliente.

```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

Essa linha cria o *socket* do cliente, denominado `clientSocket`. O primeiro parâmetro novamente indica que a rede subjacente está usando IPv4. O segundo parâmetro indica que o *socket* é do tipo `SOCK_STREAM`, ou seja, é um *socket* TCP (em vez de um UDP). Observe que de novo não estamos especificando o número de porta do *socket* cliente quando o criamos; em vez disso, deixamos que o sistema operacional o faça por nós. Agora, a próxima linha de código é muito diferente do que vimos em UDPClient:

```
clientSocket.connect((serverName,serverPort))
```

Lembre-se de que, antes de um cliente poder enviar dados ao servidor (e vice-versa) usando um *socket* TCP, primeiro deve ser estabelecida uma conexão TCP entre eles, o que é feito por meio dessa linha. O parâmetro do método `connect()` é o endereço do lado servidor da conexão. Depois que essa linha de código é executada, é feita uma apresentação de três vias e uma conexão TCP é estabelecida.

```
sentence = raw_input('Input lowercase sentence:')
```

Assim como em UDPClient, essa linha obtém uma sentença do usuário. A cadeia sentence continua a reunir caracteres até que o usuário termine a linha digitando um *Enter*. A linha de código seguinte também é muito diferente do UDPClient:

```
clientSocket.send(sentence)
```

Essa linha envia a cadeia `sentence` pelo *socket* do cliente e para a conexão TCP. Observe que o programa *não* cria um pacote explicitamente, anexando o endereço de destino ao pacote, como foi feito com os *sockets* UDP. Em vez disso, apenas deixa os bytes da cadeia sentence na conexão TCP. O cliente, então, espera para receber bytes do servidor.

```
modifiedSentence = clientSocket.recv(2048)
```

Quando os caracteres chegam do servidor, eles são colocados na cadeia `modifiedSentence`. Os caracteres continuam a ser acumulados em modifiedSentence até que a linha termine com um caractere de *Enter*. Depois de exibir a sentença em maiúsculas, fechamos o *socket* do cliente:

```
clientSocket.close()
```

Essa última linha fecha o *socket* e, portanto, fecha a conexão TCP entre cliente e servidor. Ela faz o TCP no cliente enviar uma mensagem TCP ao TCP no servidor (ver Seção 3.5).

TCPServer.py

Agora vamos examinar o programa servidor.

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
        connectionSocket, addr = serverSocket.accept()
        sentence = connectionSocket.recv(1024)
        capitalizedSentence = sentence.upper()
        connectionSocket.send(capitalizedSentence)
        connectionSocket.close()
```

Vejamos agora as linhas que diferem significativamente de UDPServer e TCPClient. Assim como em TCP-Client, o servidor cria um *socket* TCP com:

```
serverSocket=socket(AF_INET,SOCK_STREAM)
```

De modo semelhante a UDPServer, associamos o número de porta do servidor, `serverPort`, ao *socket*:

```
serverSocket.bind(('',serverPort))
```

Porém, com TCP, `serverSocket` será nosso *socket* de entrada. Depois de estabelecer essa porta de entrada, vamos esperar e ficar escutando até que algum cliente bata:

```
serverSocket.listen(1)
```

Essa linha faz com que o servidor escute as requisições de conexão TCP do cliente. O parâmetro especifica o número máximo de conexões em fila (pelo menos 1).

```
connectionSocket, addr = serverSocket.accept()
```

Quando o cliente bate a essa porta, o programa chama o método `accept()` para serverSocket, que cria um novo *socket* no servidor, chamado `connectionSocket`, dedicado a esse cliente específico. Cliente e servidor, então, completam a apresentação, criando uma conexão TCP entre o `clientSocket` do cliente e o connectionSocket do servidor. Após estabelecer a conexão TCP, cliente e servidor podem enviar bytes um para o outro por ela. Com TCP, todos os bytes enviados de um lado têm garantias não apenas de que chegarão ao outro lado, mas também na ordem.

```
connectionSocket.close()
```

Nesse programa, depois de enviar a sentença modificada ao cliente, fechamos o *socket* da conexão. Mas, como `serverSocket` permanece aberto, outro cliente agora pode bater à porta e enviar uma sentença ao servidor, para que seja modificada.

Isso conclui nossa discussão sobre programação de *sockets* em TCP. Encorajamos o leitor a executar os dois programas em dois hospedeiros separados, e também a modificá-los para realizar objetivos ligeiramente diferentes. Compare o par de programas UDP com o par de programas TCP e repare suas diferenças. Você também deverá realizar várias tarefas de programação de *sockets* descritas ao final dos Capítulos 2, 4 e 7. Por fim, esperamos que, um dia, depois de dominar estes e outros programas de *sockets* mais avançados, você escreva sua própria aplicação popular para redes, fique rico, famoso e lembre-se dos autores deste livro!

2.8 RESUMO

Neste capítulo, estudamos os aspectos conceituais e os aspectos de implementação de aplicações de rede. Conhecemos a onipresente arquitetura cliente-servidor adotada por aplicações da Internet e examinamos sua utilização nos protocolos HTTP, FTP, SMTP, POP3 e DNS. Analisamos esses importantes protocolos de camada de aplicação e suas aplicações associadas (Web, transferência de arquivos, e-mail e DNS) com algum detalhe. Conhecemos também a arquitetura P2P, cada vez mais dominante, e examinamos sua utilização em muitas aplicações. Vimos como o API *socket* pode ser usado para construir aplicações de rede. Examinamos a utilização de *sockets* para serviços de transporte fim a fim orientados a conexão (TCP) e não orientados a conexão (UDP). A primeira etapa de nossa jornada de descida pela arquitetura das camadas da rede está concluída!

Logo no começo deste livro, na Seção 1.1, demos uma definição um tanto vaga e despojada de um protocolo. Dissemos que um protocolo é "o formato e a ordem das mensagens trocadas entre duas ou mais entidades comunicantes, bem como as ações realizadas na transmissão e/ou no recebimento de uma mensagem ou outro evento". O material deste capítulo — em particular, o estudo detalhado dos protocolos HTTP, FTP, SMTP, POP3 e DNS — agregou considerável substância a essa definição. Protocolos são o conceito fundamental de redes. Nosso estudo sobre protocolos de aplicação nos deu agora a oportunidade de desenvolver uma noção mais intuitiva do que eles realmente são.

Na Seção 2.1, descrevemos os modelos de serviço que o TCP e o UDP oferecem às aplicações que os chamam. Nós os examinamos ainda mais de perto quando desenvolvemos, na Seção 2.7, aplicações simples que executam em TCP e UDP. Contudo, pouco dissemos sobre como o TCP e o UDP fornecem esses modelos de serviços. Por exemplo, sabemos que o TCP provê um serviço de dados confiável, mas ainda não mencionamos como ele o faz. No próximo capítulo, examinaremos cuidadosamente não apenas *o que* são protocolos de transporte, mas também o *como* e o *porquê* deles.

Agora que conhecemos a estrutura da aplicação da Internet e os protocolos de camada de aplicação, estamos prontos para continuar a descer a pilha de protocolos e examinar a camada de transporte no Capítulo 3.

EXERCÍCIOS DE FIXAÇÃO E PERGUNTAS

Questões de revisão do Capítulo 2

SEÇÃO 2.1

R1. Relacione cinco aplicações da Internet não proprietárias e os protocolos de camada de aplicação que elas usam.

R2. Qual é a diferença entre arquitetura de rede e arquitetura de aplicação?

R3. Para uma sessão de comunicação entre um par de processos, qual processo é o cliente e qual é o servidor?

R4. Em uma aplicação de compartilhamento de arquivos P2P, você concorda com a afirmação: "não existe nenhuma noção de lados cliente e servidor de uma sessão de comunicação"? Justifique sua resposta.

R5. Que informação é usada por um processo que está rodando em um hospedeiro para identificar um processo que está rodando em outro hospedeiro?

R6. Suponha que você queria fazer uma transação de um cliente remoto para um servidor da maneira mais rápida possível. Você usaria o UDP ou o TCP? Por quê?

R7. Com referência à Figura 2.4, vemos que nenhuma das aplicações relacionadas nela requer "sem perda de dados" e "temporização". Você consegue imaginar uma aplicação que requeira "sem perda de dados" e seja também altamente sensível ao atraso?

R8. Relacione quatro classes de serviços que um protocolo de transporte pode prover. Para cada uma, indique se o UDP ou o TCP (ou ambos) fornece tal serviço.

R9. Lembre-se de que o TCP pode ser aprimorado com o SSL para fornecer serviços de segurança processo a processo, incluindo a decodificação. O SSL opera na camada de transporte ou na camada de aplicação? Se o desenvolvedor da aplicação quer que o TCP seja aprimorado com o SSL, o que ele deve fazer?

SEÇÕES 2.2-2.5

R10. O que significa protocolo de apresentação (*handshaking protocol*)?

R11. Por que HTTP, FTP, SMTP, POP3 rodam sobre TCP e não sobre UDP?

R12. Considere um site de comércio eletrônico que quer manter um registro de compras para cada um de seus clientes. Descreva como isso pode ser feito com *cookies*.

R13. Descreva como o *cache* Web pode reduzir o atraso na recepção de um objeto requisitado. O *cache* Web reduzirá o atraso para todos os objetos requisitados por um usuário ou somente para alguns objetos? Por quê?

R14. Digite um comando Telnet em um servidor Web e envie uma mensagem de requisição com várias linhas. Inclua nessa mensagem a linha de cabeçalho `If-modified-since:` para forçar uma mensagem de resposta com a codificação de estado `304 Not Modified`.

R15. Por que se diz que o FTP envia informações de controle "fora da banda"?

R16. Suponha que Alice envie uma mensagem a Bob por meio de uma conta de e-mail da Web (como o Hotmail ou gmail), e que Bob acesse seu e-mail por seu servidor de correio usando POP3. Descreva como a mensagem vai do hospedeiro de Alice até o hospedeiro de Bob. Não se esqueça de relacionar a série de protocolos de camada de aplicação usados para movimentar a mensagem entre os dois hospedeiros.

R17. Imprima o cabeçalho de uma mensagem de e-mail que tenha recebido recentemente. Quantas linhas de cabeçalho Received: há nela? Analise cada uma.

R18. Do ponto de vista de um usuário, qual é a diferença entre o modo ler-e-apagar e o modo ler-e-guardar no POP3?

R19. É possível que o servidor Web e o servidor de correio de uma organização tenham exatamente o mesmo apelido para um nome de hospedeiro (por exemplo, foo.com)? Qual seria o tipo de RR que contém o nome de hospedeiro do servidor de correio?

R20. Examine seus e-mails recebidos e veja o cabeçalho de uma mensagem enviada de um usuário com um endereço de correio eletrônico .edu. É possível determinar, pelo cabeçalho, o endereço IP do hospedeiro do qual a mensagem foi enviada? Faça o mesmo para uma mensagem enviada de uma conta do gmail.

SEÇÃO 2.6

R21. No BitTorrent, suponha que Alice forneça blocos para Bob durante um intervalo de 30 s. Bob retornará, necessariamente, o favor e fornecerá blocos para Alice no mesmo intervalo? Por quê?

R22. Considere um novo par, Alice, que entra no BitTorrent sem possuir nenhum bloco. Sem qualquer bloco, ela não pode se tornar uma das quatro melhores exportadoras de dados para qualquer dos outros pares, visto que ela não possui nada para enviar. Então, como Alice obterá seu primeiro bloco?

R23. O que é uma rede de sobreposição? Ela inclui roteadores? O que são as arestas da rede de sobreposição?

R24. Considere um DHT com uma topologia da rede de sobreposição (ou seja, cada par rastreia todos os pares no sistema). Quais são as vantagens e desvantagens de um DHT circular (sem atalhos)?

R25. Relacione pelo menos quatro diferentes aplicações que são apropriadas naturalmente para arquiteturas P2P. (*Dica*: Distribuição de arquivo e mensagem instantânea são duas.)

SEÇÃO 2.7

R26. O servidor UDP descrito na Seção 2.7 precisava de um *socket* apenas, ao passo que o servidor TCP precisava de dois. Por quê? Se um servidor TCP tivesse de suportar *n* conexões simultâneas, cada uma de um hospedeiro cliente diferente, de quantos *sockets* precisaria?

R27. Para a aplicação cliente-servidor por TCP descrita na Seção 2.7, por que o programa servidor deve ser executado antes do programa cliente? Para a aplicação cliente-servidor por UDP, por que o programa cliente pode ser executado antes do programa servidor?

PROBLEMAS

P1. Falso ou verdadeiro?

a. Um usuário requisita uma página Web que consiste em algum texto e três imagens. Para essa página, o cliente enviará uma mensagem de requisição e receberá quatro mensagens de resposta.

b. Duas páginas Web distintas (por exemplo, `www.mit.edu/research.html` e `www.mit.edu/students.html`) podem ser enviadas pela mesma conexão persistente.

c. Com conexões não persistentes entre navegador e servidor de origem, é possível que um único segmento TCP transporte duas mensagens distintas de requisição HTTP.

d. O cabeçalho Date: na mensagem de resposta HTTP indica a última vez que o objeto da resposta foi modificado.

e. As mensagens de resposta HTTP nunca possuem um corpo de mensagem vazio.

P2. Leia o RFC 959 para FTP. Relacione todos os comandos de cliente que são suportados pelo RFC.

P3. Considere um cliente HTTP que queira obter um documento Web em um dado URL. Inicialmente, o endereço IP do servidor HTTP é desconhecido. Nesse cenário, quais protocolos de transporte e de camada de aplicação são necessários, além do HTTP?

P4. Considere a seguinte cadeia de caracteres ASCII capturada pelo Wireshark quando o navegador enviou uma mensagem HTTP GET (ou seja, o conteúdo real de uma mensagem HTTP GET). Os caracteres <*cr*><*lf*> são

retorno de carro e avanço de linha (ou seja, a cadeia de caracteres em itálico *<cr>* no texto abaixo representa o caractere único retorno de carro que estava contido, naquele momento, no cabeçalho HTTP). Responda às seguintes questões, indicando onde está a resposta na mensagem HTTP GET a seguir.

```
GET /cs453/index.html HTTP/1.1<cr><lf>Host: gai
a.cs.umass.edu<cr><lf>User-Agent: Mozilla/5.0 (
Windows;U; Windows NT 5.1; en-US; rv:1.7.2) Gec
ko/20040804 Netscape/7.2 (ax) <cr><lf>Accept:ex
t/xml, application/xml, application/xhtml+xml, text
/html;q=0.9, text/plain;q=0.8,image/png,*/*;q=0.5
<cr><lf>Accept-Language: en-us,en;q=0.5<cr><lf>Accept-
Encoding: zip,deflate<cr><lf>Accept-Charset: ISO
-8859-1,utf-8;q=0.7,*;q=0.7<cr><lf>Keep-Alive: 300<cr>
<lf>Connection:keep-alive<cr><lf><cr><lf>
```

a. Qual é a URL do documento requisitado pelo navegador?

b. Qual versão do HTTP o navegador está rodando?

c. O navegador requisita uma conexão não persistente ou persistente?

d. Qual é o endereço IP do hospedeiro no qual o navegador está rodando?

e. Que tipo de navegador inicia essa mensagem? Por que é necessário o tipo de navegador em uma mensagem de requisição HTTP?

P5. O texto a seguir mostra a resposta enviada do servidor em reação à mensagem HTTP GET na questão anterior. Responda às seguintes questões, indicando onde está a resposta na mensagem.

```
HTTP/1.1 200 OK<cr><lf>Date: Tue, 07 Mar 2008
    12:39:45GMT<cr><lf>Server: Apache/2.0.52 (Fedora)
    <cr><lf>Last-Modified: Sat, 10 Dec2005 18:27:46
    GMT<cr><lf>ETag: "526c3-f22-a88a4c80"<cr><lf>Accept-
    Ranges: bytes<cr><lf>Content-Length: 3874<cr><lf>
    Keep-Alive: timeout=max=100<cr><lf>Connection:
    Keep-Alive<cr><lf>Content-Type: text/html; charset=
    ISO-8859-1<cr><lf><cr><lf><!doctype html public "-
    //w3c//dtd html 4.0 transitional//en"><lf><html><lf>
    <head><lf> <meta http-equiv="Content-Type"
    content="text/html; charset=iso-8859-1"><lf> <meta
    name="GENERATOR" content="Mozilla/4.79 [en] (Windows NT
    5.0; U) Netscape]"><lf> <title>CMPSCI 453 / 591 /
    NTU-ST550A Spring 2005 homepage</title><lf></head><lf>
    <muito mais texto do documento em seguida (não mostrado)>
```

a. O servidor foi capaz de encontrar o documento com sucesso ou não? A que horas foi apresentada a resposta do documento?

b. Quando o documento foi modificado pela última vez?

c. Quantos bytes existem no documento que está retornando?

d. Quais são os 5 primeiros bytes do documento que está retornando? O servidor aceitou uma conexão persistente?

P6. Obtenha a especificação HTTP/1.1 (RFC 2616). Responda às seguintes perguntas:

a. Explique o mecanismo de sinalização que cliente e servidor utilizam para indicar que uma conexão persistente está sendo fechada. O cliente, o servidor, ou ambos, podem sinalizar o encerramento de uma conexão?

b. Que serviços de criptografia são providos pelo HTTP?

c. O cliente é capaz de abrir três ou mais conexões simultâneas com um determinado servidor?

d. Um servidor ou um cliente pode abrir uma conexão de transporte entre eles se um dos dois descobrir que a conexão ficou lenta por um tempo. É possível que um lado comece a encerrar a conexão enquanto o outro está transmitindo dados por meio dessa conexão? Explique.

P7. Suponha que você clique com seu navegador Web sobre um ponteiro para obter uma página e que o endereço IP para o URL associado não esteja no *cache* de seu hospedeiro local. Portanto, será necessária uma consulta ao DNS para obter o endereço IP. Considere que n servidores DNS sejam visitados antes que seu hospedeiro receba o endereço IP do DNS; as visitas sucessivas incorrem em um RTT igual a $RTT_1, \ldots, RTT_n$. Suponha ainda que a página associada ao ponteiro contenha exatamente um objeto que consiste em uma pequena quantidade de texto HTML. Seja RTT_0 o RTT entre o hospedeiro local e o servidor que contém o objeto. Admitindo que o tempo de transmissão seja zero, quanto tempo passará desde que o cliente clica o ponteiro até que receba o objeto?

P8. Com referência ao Problema 7, suponha que o arquivo HTML referencie oito objetos muito pequenos no mesmo servidor. Desprezando tempos de transmissão, quanto tempo passa, usando-se:

a. HTTP não persistente sem conexões TCP paralelas?

b. HTTP não persistente com o navegador configurado para 5 conexões paralelas?

c. HTTP persistente?

P9. Considere a Figura 2.12, que mostra uma rede institucional conectada à Internet. Suponha que o tamanho médio do objeto seja 850 mil bits e que a taxa média de requisição dos navegadores da instituição aos servidores de origem seja 16 requisições por segundo. Suponha também que a quantidade de tempo que leva desde o instante em que o roteador do lado da Internet do enlace de acesso transmite uma requisição HTTP até que receba a resposta seja 3 segundos em média (veja Seção 2.2.5). Modele o tempo total médio de resposta como a soma do atraso de acesso médio (isto é, o atraso entre o roteador da Internet e o roteador da instituição) e o tempo médio de atraso da Internet. Para a média de atraso de acesso, use $\Delta(1 - \Delta\beta)$, sendo Δ o tempo médio requerido para enviar um objeto pelo enlace de acesso e β a taxa de chegada de objetos ao enlace de acesso.

a. Determine o tempo total médio de resposta.

b. Agora, considere que um *cache* é instalado na LAN institucional e que a taxa de resposta local seja 0,4. Determine o tempo total de resposta.

P10. Considere um enlace curto de 10 m através do qual um remetente pode transmitir a uma taxa de 150 bits/s em ambas as direções. Suponha que os pacotes com dados tenham 100 mil bits de comprimento, e os pacotes que contêm controle (por exemplo, ACK ou apresentação) tenham 200 bits de comprimento. Admita que N conexões paralelas recebam cada $1/N$ da largura de banda do enlace. Agora, considere o protocolo HTTP e suponha que cada objeto baixado tenha 100 Kbits de comprimento e que o objeto inicial baixado contenha 10 objetos referenciados do mesmo remetente. Os *downloads* paralelos por meio de instâncias paralelas de HTTP não persistente fazem sentido nesse caso? Agora considere o HTTP persistente. Você espera ganhos significativos sobre o caso não persistente? Justifique sua resposta.

P11. Considere o cenário apresentado na questão anterior. Agora suponha que o enlace é compartilhado por Bob e mais quatro usuários. Bob usa instâncias paralelas de HTTP não persistente, e os outros quatro usam HTTP não persistente sem *downloads* paralelos.

a. As conexões paralelas de Bob o ajudam a acessar páginas Web mais rapidamente? Por quê? Por que não?

b. Se cinco usuários abrirem cinco instâncias paralelas de HTTP não persistente, então as conexões paralelas de Bob ainda seriam úteis? Por quê? Por que não?

P12. Escreva um programa TCP simples para um servidor que aceite linhas de entrada de um cliente e envie as linhas para a saída-padrão do servidor. (Você pode fazer isso modificando o programa TCPServer.py no texto.) Compile e execute seu programa. Em qualquer outra máquina que contenha um navegador Web, defina o servidor *proxy* no navegador para a máquina que está executando seu programa servidor e também configure o número de porta adequadamente. Seu navegador deverá agora enviar suas mensagens de requisição GET a seu servidor, e este deverá apresentar as mensagens em sua saída-padrão. Use essa plataforma para determinar se seu navegador gera mensagens GET condicionais para objetos que estão em *caches* locais.

P13. Qual é a diferença entre `MAIL FROM:` em SMTP e `FROM:` na própria mensagem de correio?

P14. Como o SMTP marca o final de um corpo de mensagem? E o HTTP? O HTTP pode usar o mesmo método que o SMTP para marcar o fim de um corpo de mensagem? Explique.

P15. Leia o RFC 5321 para SMTP. O que significa MTA? Considere a seguinte mensagem *spam* recebida (modificada de um *spam* verdadeiro). Admitindo que o criador desse *spam* seja malicioso e que todos os outros hospedeiros sejam honestos, identifique o hospedeiro malicioso que criou essa mensagem spam.

```
From - Fri Nov 07 13:41:30 2008
Return-Path: <tennis5@pp33head.com>
Received: from barmail.cs.umass.edu
(barmail.cs.umass.edu [128.119.240.3]) by cs.umass.edu
(8.13.1/8.12.6) for <hg@cs.umass.edu>; Fri, 7 Nov 2008
13:27:10 -0500
Received: from asusus-4b96 (localhost [127.0.0.1]) by
barmail.cs.umass.edu (Spam Firewall) for
<hg@cs.umass.edu>; Fri, 7 Nov 2008 13:27:07 -0500
(EST)
Received: from asusus-4b96 ([58.88.21.177]) by
barmail.cs.umass.edu for <hg@cs.umass.edu>; Fri,
07 Nov 2008 13:27:07 -0500 (EST)
Received: from [58.88.21.177] by
inbnd55.exchangeddd.com; Sat, 8 Nov 2008 01:27:07 +0700
From: "Jonny" <tennis5@pp33head.com>
To: <hg@cs.umass.edu>
Subject: How to secure your savings
```

P16. Leia o RFC do POP3 [RFC 1939]. Qual é a finalidade do comando UIDL do POP3?

P17. Imagine que você acesse seu e-mail com POP3.

a. Suponha que você configure seu cliente de correio POP para funcionar no modo ler-e-apagar. Conclua a seguinte transação:

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: blah blah ...
S: ..........blah
S: .
?
?
```

b. Suponha que você configure seu cliente de correio POP para funcionar no modo ler-e-guardar. Conclua a seguinte transação:

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: blah blah ...
S: ..........blah
S: .
?
?
```

c. Suponha que você configure seu cliente de correio POP para funcionar no modo ler-e-guardar. Usando sua solução no item (b), considere que você recupere as mensagens 1 e 2, saia do POP e então, 5 minutos mais tarde, acesse outra vez o POP para obter um novo e-mail. Imagine que nenhuma outra mensagem foi enviada nesse intervalo de 5 minutos. Elabore uma transcrição dessa segunda sessão POP.

P18. a. O que é um banco de dados *whois*?

b. Use vários bancos de dados *whois* da Internet para obter os nomes de dois servidores DNS. Cite quais bancos de dados *whois* você utilizou.

c. Use *nslookup* em seu hospedeiro local para enviar consultas DNS a três servidores DNS: seu servidor DNS local e os dois servidores DNS que encontrou na parte (b). Tente consultar registros dos tipos A, NS e MX. Faça um resumo do que encontrou.

d. Use *nslookup* para encontrar um servidor Web que tenha vários endereços IP. O servidor de sua instituição (escola ou empresa) tem vários endereços IP?

e. Use o banco de dados *whois* ARIN para determinar a faixa de endereços IP usados por sua universidade.

f. Descreva como um invasor pode usar bancos de dados *whois* e a ferramenta *nslookup* para fazer o reconhecimento de uma instituição antes de lançar um ataque.

g. Discuta por que bancos de dados *whois* devem estar disponíveis publicamente.

P19. Neste problema, utilizamos a ferramenta funcional *dig* disponível em hospedeiros Unix e Linux para explorar a hierarquia dos servidores DNS. Lembre de que, na Figura 2.21, um servidor DNS de nível superior na hierarquia do DNS delega uma consulta DNS para um servidor DNS de nível inferior na hierarquia, enviando de volta ao cliente DNS o nome daquele servidor DNS de nível inferior. Primeiro, leia a *man page* sobre a ferramenta *dig* e responda às seguintes questões:

a. Iniciando com o servidor DNS raiz (de um dos servidores raiz [a-m].root-servers.net), construa uma sequência de consultas para o endereço IP para seu servidor de departamento utilizando o *dig*. Mostre a relação de nomes de servidores DNS na cadeia de delegação ao responder à sua consulta.

b. Repita o item (a) com vários sites da Internet populares, como google.com, yahoo.com ou amazon.com.

P20. Suponha que você consiga acessar os *caches* nos servidores DNS locais do seu departamento. Você é capaz de propor uma maneira de determinar, em linhas gerais, os servidores (fora de seu departamento) que são mais populares entre os usuários do seu departamento? Explique.

P21. Suponha que seu departamento possua um servidor DNS local para todos os computadores do departamento. Você é um usuário comum (ou seja, não é um administrador de rede/sistema). Você consegue encontrar um modo de determinar se um site da Internet externo foi muito provavelmente acessado de um computador do seu departamento alguns segundos atrás? Explique.

P22. Considere um arquivo de distribuição de $F = 15$ Gbits para N pares. O servidor possui uma taxa de *upload* de $u_s = 30$ Mbits/s e cada par possui uma taxa de download de $d_i = 2$ Mbits/s e uma taxa de *upload* de u. Para $N = 10$, 100 e 1.000 e $u = 300$ Kbits/s, 700 Kbits/s e 2 Mbits/s, prepare um gráfico apresentando o tempo mínimo de distribuição para cada uma das combinações de N e u para o modo cliente-servidor e para o modo distribuição P2P.

P23. Considere distribuir um arquivo de F bits para N pares utilizando uma arquitetura cliente-servidor. Admita um modelo fluido no qual o servidor pode transmitir de modo simultâneo para diversos pares, a diferentes taxas, desde que a taxa combinada não ultrapasse u_s.

a. Suponha que $u_s/N \leq d_{min}$. Especifique um esquema de distribuição que possua o tempo de distribuição de NF/u_s.

b. Suponha que $u_s/N \geq d_{min}$. Especifique um esquema de distribuição que possua o tempo de distribuição de F/d_{min}.

c. Conclua que o tempo mínimo de distribuição é, geralmente, dado por $\max\{NF/u_s, F/d_{min}\}$.

P24. Considere distribuir um arquivo de F bits para N pares utilizando uma arquitetura P2P. Admita um modelo fluido e que d_{min} é muito grande, de modo que a largura de banda do *download* do par nunca é um gargalo.

a. Suponha que $u_s \leq (u_s + u_1 + ... + u_N)/N$. Especifique um esquema de distribuição que possua o tempo de distribuição de F/u_s.

b. Suponha que $u_s \geq (u_s + u_1 + ... + u_N)/N$. Especifique um esquema de distribuição que possua o tempo de distribuição de $NF/(u_s + u_1 + ... + u_N)$.

c. Conclua que o tempo mínimo de distribuição é, em geral, dado por $\max\{F/u_s, NF/(u_s + u_1 + ... + u_N)\}$.

P25. Considere uma rede de sobreposição com N pares ativos, em que cada dupla de pares possua uma conexão TCP. Além disso, suponha que as conexões TCP passem por um total de M roteadores. Quantos nós e arestas há na rede de sobreposição correspondente?

P26. Suponha que Bob tenha entrado no BitTorrent, mas ele não quer fazer o *upload* de nenhum dado para qualquer outro par (denominado carona).

a. Bob alega que consegue receber uma cópia completa do arquivo compartilhado pelo grupo. A alegação de Bob é possível? Por quê?

b. Bob alega ainda que ele pode "pegar carona" de um modo mais eficiente usando um conjunto de diversos computadores (com endereços IP distintos) no laboratório de informática de seu departamento. Como ele pode fazer isso?

P27. No exemplo de DHT circular na Seção 2.6.2, suponha que o par 3 descobriu que o par 5 saiu. Como o par 3 atualiza informações sobre o estado de seu sucessor? Qual par é agora seu primeiro sucessor? E seu segundo sucessor?

P28. No exemplo de DHT circular na Seção 2.6.2, suponha que um novo par 6 queira se juntar ao DHT e inicialmente só conheça o endereço IP do par 15. Que etapas são tomadas?

P29. Como um número inteiro em $[0,2^n - 1]$ pode ser expresso como um número binário de n bit em um DHT, cada chave pode ser expressa como $k = (k_0, k_1, ..., k_{n-1})$, e cada identificador de par pode ser expresso como $p = (p_0, p_1, ..., p_{n-1})$. Vamos, agora, definir a distância XOR entre a chave k e o par p como

$$d(k, p) = \sum_{j=0}^{n-1} |k_j - p_j| 2^j$$

Descreva como essa métrica pode ser usada para designar duplas (chave, valor) a pares. (Para aprender mais sobre como construir um DHT eficiente usando essa métrica natural, consulte Maymounkov [2002], no qual o DHT Kademlia é descrito.)

P30. Como os DHTs são redes de sobreposição, eles nem sempre se adequam bem à rede física de sobreposição considerando que dois pares vizinhos podem estar fisicamente muito distantes; por exemplo, um par poderia estar na Ásia e seu vizinho, na América do Norte. Se atribuirmos identificadores de maneira aleatória e uniforme para pares recém-unidos, esse esquema de atribuição causaria essa incompatibilidade? Explique. E como tal incompatibilidade afetaria o desempenho do DHT?

P31. Instale e compile os programas Python TCPClient e UDPClient em um hospedeiro e TCPServer e UDPServer em outro.

a. Suponha que você execute TCPClient antes de executar TCPServer. O que acontece? Por quê?

b. Imagine que você execute UDPClient antes de UDPServer. O que acontece? Por quê?

c. O que acontece se você usar números de porta diferentes para os lados cliente e servidor?

P32. Suponha que, em UDPClient.py, depois de criarmos o *socket*, acrescentemos a linha:

```
clientSocket.bind(('', 5432))
```

Será necessário mudar UDPServer.py? Quais são os números de porta para os *sockets* em UDPClient e UDPServer? Quais eram esses números antes dessa mudança?

P33. Você consegue configurar seu navegador para abrir várias conexões simultâneas com um site? Quais são as vantagens e desvantagens de ter um grande número de conexões TCP simultâneas?

P34 Vimos que os *sockets* TCP da Internet tratam os dados sendo enviados como um fluxo de bytes, mas *sockets* UDP reconhecem limites de mensagem. Quais são uma vantagem e uma desvantagem da API orientada a byte em relação a fazer com que a API reconheça e preserve explicitamente os limites de mensagem definidos pela aplicação?

P35. O que é o servidor Web Apache? Quanto ele custa? Que funcionalidade ele possui atualmente? Você pode querer ver na Wikipedia para responder a essa pergunta.

P36. Muitos clientes BitTorrent utilizam DHTs para criar um rastreador distribuído. Para esses DHTs, qual é a "chave" e qual é o "valor"?

TAREFAS DE PROGRAMAÇÃO DE *SOCKETS*

O site de apoio deste livro inclui seis tarefas de programação de *sockets*. As quatro primeiras são resumidas a seguir. A quinta utiliza o protocolo ICMP e está resumida ao final do Capítulo 4. A sexta tarefa emprega protocolos de multimídia e está resumida no final do Capítulo 7. Recomenda-se bastante que os alunos completem várias, se não todas essas tarefas. Os alunos podem achar detalhes completos dessas tarefas, bem como trechos importantes do código Python, no site http://www.awl.com/kurose-ross.

Tarefa 1: Servidor Web

Nesta tarefa, você desenvolverá um servidor Web simples em Python, capaz de processar apenas uma requisição. Seu servidor Web (i) criará um *socket* de conexão quando contatado por um cliente (navegador); (ii) receberá a requisição HTTP dessa conexão; (iii) analisará a requisição para determinar o arquivo específico sendo requisitado; (iv) obterá o arquivo requisitado do sistema de arquivo do servidor; (v) criará uma mensagem de resposta HTTP consistindo no arquivo requisitado precedido por linhas de cabeçalho; e (vi) enviará a resposta pela conexão TCP ao navegador requisitante. Se um navegador requisitar um arquivo que não está presente no seu servidor, seu servidor deverá retornar uma mensagem de erro "404 Not Found".

No site de apoio, oferecemos o código estrutural para o seu servidor. Sua tarefa é concluir o código, rodar seu servidor e depois testá-lo enviando requisições de navegadores rodando em hospedeiros diferentes. Se você rodar seu servidor em um hospedeiro que já tem um servidor Web rodando nele, então deverá usar uma porta diferente da porta 80 para o seu servidor.

Tarefa 2: UDP Pinger

Nesta tarefa de programação, você escreverá um programa *ping* do cliente em Python. Seu cliente enviará uma mensagem *ping* simples a um servidor, receberá uma mensagem *pong* correspondente de volta do servidor e determinará o atraso entre o momento em que o cliente enviou a mensagem *ping* e recebeu a mensagem *pong*. Esse atraso é denominado tempo de viagem de ida e volta (*round-trip time* — RTT). A funcionalidade oferecida pelo cliente e servidor é semelhante à fornecida pelo programa *ping* padrão, disponível nos sistemas operacionais modernos. Porém, os programas *ping* padrão usam o Internet Control Message Protocol (ICMP) (que veremos no Capítulo 4). Aqui, criaremos um programa *ping* baseado em UDP, fora do padrão (porém simples!).

Seu programa *ping* deverá enviar 10 mensagens *ping* ao servidor de destino por meio de UDP. Para cada mensagem, seu cliente deverá determinar e imprimir o RTT quando a mensagem *pong* correspondente for retornada. Como o UDP é um protocolo não confiável, um pacote enviado pelo cliente ou servidor poderá ser perdido. Por esse motivo, o cliente não poderá esperar indefinidamente por uma resposta a uma mensagem *ping*. Você deverá fazer que o cliente espere até 1 s por uma resposta do servidor; se nenhuma resposta for recebida, o cliente deverá considerar que o pacote foi perdido e imprimir uma mensagem de acordo.

Nesta tarefa, você receberá o código completo para o servidor (disponível no site de apoio). Sua tarefa é escrever o código cliente, que será semelhante ao código do servidor. Recomendamos que, primeiro, você estude cuidadosamente o código do servidor. Depois, poderá escrever seu código cliente, cortando e colando à vontade as linhas do código do servidor.

Tarefa 3: Cliente de correio

O objetivo desta tarefa de programação é criar um cliente de correio simples, que envia e-mail a qualquer destinatário. Seu cliente precisará estabelecer uma conexão TCP com um servidor de correio (por exemplo, um servidor de correio do Google), dialogar com esse servidor usando o protocolo SMTP, enviar uma mensagem de correio a um destinatário (por exemplo, seu amigo) pelo servidor de correio e, por fim, fechar a conexão TCP com o servidor de correio.

Para esta tarefa, o site de apoio oferece o código estrutural para o seu cliente. Sua tarefa é completar o código e testar seu cliente, enviando e-mail para contas de usuário diferentes. Você também pode tentar enviar por diferentes servidores (por exemplo, por um servidor de correio do Google e pelo servidor de correio da sua universidade).

Tarefa 4: Servidor *proxy* Web *multithreaded*

Nesta tarefa, você desenvolverá um *proxy* da Web. Quando seu *proxy* receber de um navegador uma requisição HTTP para um objeto, ele gerará uma nova requisição HTTP para o mesmo objeto e a enviará para o servidor de origem. Quando o *proxy* receber a resposta HTTP correspondente com o objeto do servidor de origem, ele criará uma nova resposta HTTP, incluindo o objeto, e a enviará ao cliente. Esse *proxy* será *multithreaded*, de modo que poderá lidar com várias requisições ao mesmo tempo.

Para esta tarefa, o site de apoio oferece o código estrutural para o servidor *proxy*. Seu trabalho é completar o código e depois testá-lo fazendo diferentes navegadores requisitarem objetos Web por meio do seu *proxy*.

WIRESHARK LAB: HTTP

Depois de ter molhado nossos pés com o analisador de pacotes Wireshark no Laboratório 1, agora estamos prontos para usar o Wireshark para investigar protocolos em operação. Neste laboratório, vamos explorar diversos aspectos do protocolo HTTP: a interação básica GET/resposta, formatos de mensagem HTTP, recuperação de grandes arquivos HTML, recuperação de arquivos HTML com URLs embutidas, conexões persistentes e não persistentes, e autenticação e segurança HTTP.

Como acontece com todos os laboratórios Wireshark, o desenvolvimento completo está disponível no site de apoio do livro.

WIRESHARK LAB: DNS

Neste laboratório, examinamos mais de perto o lado cliente do DNS, o protocolo que traduz nomes de hospedeiro da Internet em endereços IP. Lembre-se de que, na Seção 2.5, vimos que o papel do cliente no DNS é bastante simples — um cliente envia uma consulta ao seu servidor DNS local e recebe uma resposta de volta. Muita coisa pode acontecer debaixo dos panos, invisível aos clientes DNS, à medida que servidores DNS hierárquicos se comunicam entre si para resolver, de forma recursiva ou iterativa, a consulta DNS do cliente. Porém, do ponto de vista do cliente DNS, o protocolo é bem simples — uma consulta é formulada ao servidor DNS local e uma resposta é recebida desse servidor. Observamos o DNS em ação neste laboratório.

Como acontece com todos os Wireshark labs, a descrição completa está disponível no site de apoio do livro.

ENTREVISTA

Marc Andreessen

Marc Andreessen é um dos criadores do Mosaic, o navegador que tornou a World Wide Web popular em 1993. O Mosaic tinha uma interface limpa, facilmente entendida, e foi o primeiro navegador a exibir imagens em linha com texto. Em 1994, Marc Andreessen e Jim Clark fundaram a Netscape, cujo navegador foi, de longe, o mais popular durante meados da década de 1990. A Netscape também desenvolveu o protocolo Secure *Sockets* Layer (SSL) e muitos produtos de servidor da Internet, incluindo os servidores de correio e servidores Web baseados em SSL. Agora, ele é um dos fundadores e sócio-geral da empresa de empreendimento de risco Andreessen Horowitz, supervisionando o desenvolvimento de portfólio com *holdings* como Facebook, Foursquare, Groupon, Jawbone, Twitter e Zynga. Ele atua em diversos comitês, incluindo Bump, eBay, Glam Media, Facebook e Hewlett-Packard. Possui bacharelado em Ciência da Computação pela University of Illinois em Urbana-Champaign.

Como você se interessou por computação? Você sempre soube que queria trabalhar em tecnologia da informação?

As revoluções do videogame e da computação pessoal chegaram exatamente quando eu estava crescendo — a computação pessoal era a nova fronteira da tecnologia no final dos anos 1970 e início dos 1980. E não foi apenas Apple e o IBM PC, mas também centenas de novas empresas como Commodore e Atari. Eu aprendi a programar por um livro denominado "Instant Freeze-Dried BASIC" (BASIC instantâneo congelado a vácuo), com dez anos, e tive meu primeiro computador (um TRS-80 Color Computer — pode procurar!) com doze anos.

Por favor, descreva um ou dois dos projetos mais interessantes em que você já trabalhou durante sua carreira. Quais foram os maiores desafios?

Sem dúvida alguma, o projeto mais interessante foi o navegador Web Mosaic original, em 1992-1993; e o maior desafio foi fazer que alguém o levasse a sério na época, quando todos pensavam que o futuro interativo seria entregue na forma de "televisão interativa" por empresas imensas, não como a Internet por iniciantes.

O que o estimula a respeito do futuro das redes e da Internet? Quais são suas maiores preocupações?

A coisa mais interessante é a imensa fronteira inexplorada de aplicações e serviços que programadores e empreendedores são capazes de explorar — a Internet deslanchou a criatividade a um nível que, acredito eu, nunca vimos antes. Minha maior preocupação é o princípio das consequências não intencionadas — nem sempre sabemos as implicações do que fazemos, como a Internet sendo usada por governos para realizar um novo nível de vigilâncias sobre os cidadãos.

Há algo em particular que os estudantes deverão saber à medida que a tecnologia Web avança?

A velocidade da mudança — a coisa mais importante a aprender é como aprender — como adaptar-se de modo flexível às mudanças nas tecnologias específicas e como manter uma mente aberta sobre as novas oportunidades e possibilidades enquanto você prossegue em sua carreira.

Que pessoas o inspiraram profissionalmente?

Vannevar Bush, Ted Nelson, Doug Engelbart, Nolan Bushnell, Bill Hewlett e Dave Packard, Ken Olsen, Steve Jobs, Steve Wozniak, Andy Grove, Grace Hopper, Hedy Lamarr, Alan Turing, Richard Stallman.

Quais são suas recomendações para estudantes que desejam seguir carreira em computação e tecnologia da informação?

Aprofunde-se ao máximo possível para entender como a tecnologia é criada, e depois, complemente com o aprendizado de como o negócio funciona.

A tecnologia pode resolver os problemas do mundo?

Não, mas avançamos o padrão de vida das pessoas com o crescimento econômico, e quase todo crescimento econômico no decorrer da história veio da tecnologia — portanto, melhor que isso é impossível.

C A P Í T U L O

1 2 3 4 5 6 7 8 9

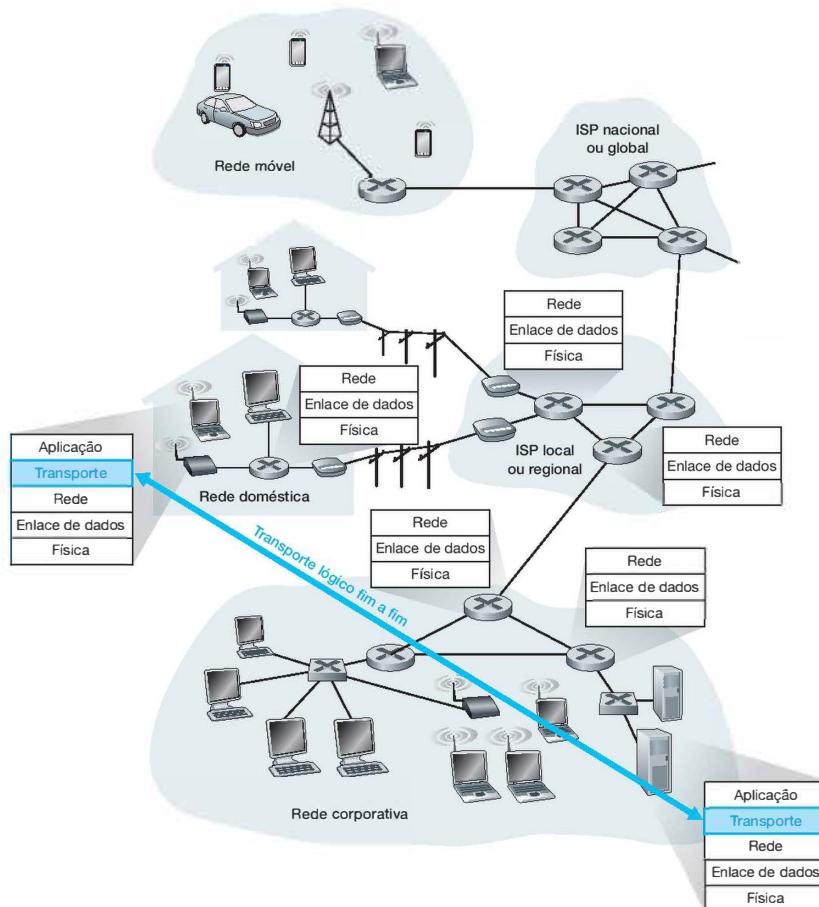
CAMADA DE TRANSPORTE

Posicionada entre a de aplicação e a de rede, a camada de transporte é uma peça central da arquitetura de rede em camadas. Ela desempenha o papel fundamental de fornecer serviços de comunicação diretamente aos processos de aplicação que rodam em hospedeiros diferentes. A abordagem pedagógica que adotamos neste capítulo é alternar entre discussões de princípios de camada de transporte e o modo como tais princípios são colocados em prática em protocolos existentes; como de costume, daremos particular ênfase aos protocolos da Internet, em especial aos de camada de transporte TCP e UDP.

Começaremos discutindo a relação entre as camadas de transporte e de rede, preparando o cenário para o exame de sua primeira função importante — ampliar o serviço de entrega da camada de rede entre dois sistemas finais para um serviço de entrega entre dois processos da camada de aplicação que rodam nos sistemas finais. Ilustraremos essa função quando abordarmos o UDP, o protocolo de transporte não orientado para conexão da Internet.

Depois, retornaremos aos princípios e trataremos de um dos problemas mais fundamentais de redes de computadores — como duas entidades podem se comunicar de maneira confiável por um meio que pode perder e corromper dados. Mediante uma série de cenários cada vez mais complicados (e realistas!), construiremos um conjunto de técnicas que os protocolos de transporte utilizam para resolver esse problema. Então, mostraremos como esses princípios estão incorporados no TCP, o protocolo de transporte orientado para conexão da Internet.

Em seguida, passaremos para um segundo problema fundamentalmente importante em redes — o controle da taxa de transmissão de entidades de camada de transporte para evitar ou se recuperar de congestionamentos dentro da rede. Consideraremos as causas e consequências do congestionamento, bem como técnicas de controle de congestionamento comumente usadas. Após adquirir um sólido conhecimento das questões que estão por trás do controle de congestionamento, estudaremos seu tratamento pelo TCP.

3.1 INTRODUÇÃO E SERVIÇOS DE CAMADA DE TRANSPORTE

Nos dois capítulos anteriores, citamos o papel da camada de transporte e os serviços que ela fornece. Vamos revisar rapidamente o que já aprendemos sobre a camada de transporte.

Um protocolo da camada de transporte fornece **comunicação lógica** entre processos de aplicação que rodam em hospedeiros diferentes. *Comunicação lógica* nesse contexto significa que, do ponto de vista de uma aplicação, tudo se passa como se os hospedeiros que rodam os processos estivessem conectados diretamente;

na verdade, eles poderão estar em lados opostos do planeta, conectados por diversos roteadores e uma ampla variedade de tipos de enlace. Processos de aplicação usam a comunicação lógica fornecida pela camada de transporte para enviar mensagens entre si, livres da preocupação dos detalhes da infraestrutura física utilizada para transportá-las. A Figura 3.1 ilustra a noção de comunicação lógica.

Como vemos na Figura 3.1, protocolos da camada de transporte são implementados nos sistemas finais, mas não em roteadores de rede. No lado remetente, a camada de transporte converte as mensagens que recebe de um processo de aplicação remetente em pacotes de camada de transporte, denominados **segmentos de camada de transporte** na terminologia da Internet. Isso é (possivelmente) feito fragmentando-se as mensagens da aplicação em pedaços menores e adicionando-se um cabeçalho de camada de transporte a cada pedaço para criar o segmento de camada de transporte. Essa camada, então, passa o segmento para a de rede no sistema final remetente, onde ele é encapsulado em um pacote de camada de rede (um datagrama) e enviado ao destinatário. É importante notar que roteadores de rede agem somente nos campos de camada de rede do datagrama; isto é, não examinam os campos do segmento de camada de transporte encapsulado com o datagrama. No lado destinatário, a camada

FIGURA 3.1 A CAMADA DE TRANSPORTE FORNECE COMUNICAÇÃO LÓGICA, E NÃO FÍSICA, ENTRE PROCESSOS DE APLICAÇÕES

de rede extrai do datagrama o segmento de camada de transporte e passa-o para a camada de transporte que, em seguida, processa o segmento recebido, disponibilizando os dados para a aplicação destinatária.

Mais de um protocolo de camada de transporte poderão estar disponíveis às aplicações de rede. Por exemplo, a Internet possui dois protocolos — TCP e UDP. Cada um oferece um conjunto diferente de serviços de camada de transporte à aplicação chamadora.

3.1.1 Relação entre as camadas de transporte e de rede

Lembre-se de que a camada de transporte se situa logo acima da de rede na pilha de protocolos. Enquanto um protocolo de camada de transporte fornece comunicação lógica entre *processos* que rodam em hospedeiros diferentes, um protocolo de camada de rede fornece comunicação lógica entre *hospedeiros*. A distinção é sutil, mas importante. Vamos examiná-la com o auxílio de uma analogia com moradias.

Considere duas casas, uma na Costa Leste e outra na Costa Oeste dos Estados Unidos, cada qual com uma dúzia de crianças. As crianças da Costa Leste são primas das crianças da Costa Oeste e todas adoram escrever cartas umas para as outras — cada criança escreve a cada primo uma vez por semana e cada carta é entregue pelo serviço de correio tradicional dentro de um envelope separado. Assim, uma casa envia 144 cartas por semana para a outra. (Essas crianças economizariam muito dinheiro se tivessem e-mail!) Em cada moradia há uma criança responsável pela coleta e distribuição da correspondência — Ann, na casa da Costa Oeste, e Bill, na da Costa Leste. Toda semana, Ann coleta a correspondência de seus irmãos e irmãs e a coloca no correio. Quando as cartas chegam à casa da Costa Oeste, também é Ann quem tem a tarefa de distribuir a correspondência, trazida pelo carteiro, a seus irmãos e irmãs. Bill realiza o mesmo trabalho na casa da Costa Leste.

Neste exemplo, o serviço postal oferece uma comunicação lógica entre as duas casas — ele movimenta a correspondência de uma residência para outra, e não de uma pessoa para outra. Por outro lado, Ann e Bill oferecem comunicação lógica entre os primos — eles coletam e entregam a correspondência de seus irmãos e irmãs. Note que, do ponto de vista dos primos, Ann e Bill *são* o serviço postal, embora sejam apenas uma parte (a parte do sistema final) do processo de entrega fim a fim. Esse exemplo das moradias é uma analogia interessante para explicar como a camada de transporte se relaciona com a camada de rede:

mensagens de aplicação = cartas em envelopes
processos = primos
hospedeiros (também denominados sistemas finais) = casas
protocolo de camada de transporte = Ann e Bill
protocolo de camada de rede = serviço postal (incluindo os carteiros)

Continuando com essa analogia, observe que Ann e Bill fazem todo o trabalho dentro de suas respectivas casas; eles não estão envolvidos, por exemplo, com a classificação da correspondência em nenhuma central intermediária dos correios ou com o transporte de uma central a outra. De maneira semelhante, protocolos de camada de transporte moram nos sistemas finais, onde movimentam mensagens de processos de aplicação para a borda da rede (isto é, para a camada de rede) e vice-versa, mas não interferem no modo como as mensagens são movimentadas dentro do núcleo. Na verdade, como ilustrado na Figura 3.1, roteadores intermediários não atuam sobre (nem reconhecem) qualquer informação que a camada de transporte possa ter anexado às mensagens da aplicação.

Prosseguindo com nossa saga familiar, suponha agora que, quando Ann e Bill saem de férias, outro par de primos — digamos, Susan e Harvey — substitua-os e encarregue-se da coleta interna da correspondência e de sua entrega. Infelizmente para as duas famílias, eles não desempenham essa tarefa do mesmo modo que Ann e Bill. Por serem crianças mais novas, Susan e Harvey recolhem e entregam a correspondência com menos frequência e, às vezes, perdem cartas (que acabam mastigadas pelo cão da família). Assim, o par de primos Susan e Harvey não oferece o mesmo conjunto de serviços (isto é, o mesmo modelo de serviço) proporcionado por Ann e Bill. De modo similar, uma rede de computadores pode disponibilizar vários protocolos de transporte, em que cada um oferece um modelo de serviço diferente às aplicações.

Os serviços que Ann e Bill podem fornecer são claramente limitados pelos possíveis serviços que os correios fornecem. Por exemplo, se o serviço postal não estipula um prazo máximo para entregar a correspondência entre as duas casas (digamos, três dias), então não há nenhuma possibilidade de Ann e Bill definirem um atraso máximo para a entrega da correspondência entre qualquer par de primos. De maneira semelhante, os serviços que um protocolo de transporte pode fornecer são muitas vezes limitados pelo modelo de serviço do protocolo subjacente da camada de rede. Se o protocolo de camada de rede não puder dar garantias contra atraso ou garantias de largura de banda para segmentos de camada de transporte enviados entre hospedeiros, então o protocolo de camada de transporte não poderá dar essas mesmas garantias para mensagens de aplicação enviadas entre processos.

No entanto, certos serviços *podem* ser oferecidos por um protocolo de transporte mesmo quando o protocolo de rede subjacente não oferece o serviço correspondente na camada de rede. Por exemplo, como veremos neste capítulo, um protocolo de transporte pode oferecer serviço confiável de transferência de dados a uma aplicação mesmo quando o protocolo subjacente da rede não é confiável, isto é, mesmo quando o protocolo de rede perde, embaralha ou duplica pacotes. Como outro exemplo (que exploraremos no Capítulo 8, quando discutirmos segurança de rede), um protocolo de transporte pode usar criptografia para garantir que as mensagens da aplicação não sejam lidas por intrusos mesmo quando a camada de rede não puder garantir o sigilo de segmentos de camada de transporte.

3.1.2 Visão geral da camada de transporte na Internet

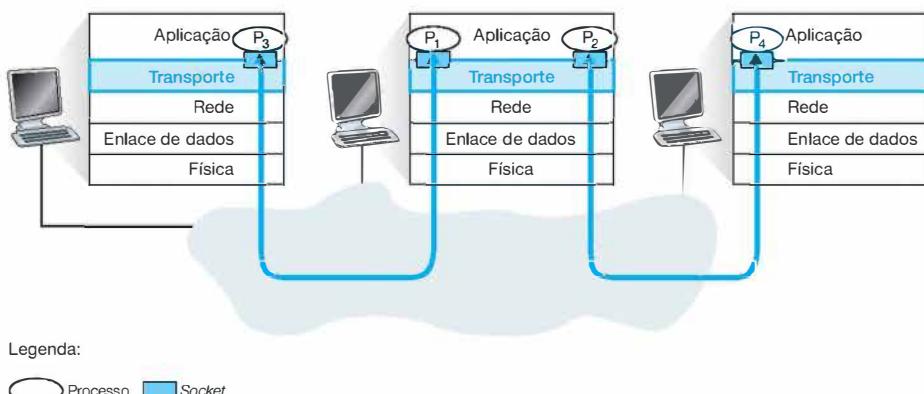
Lembre-se de que a Internet — e, de maneira mais geral, a rede TCP/IP — disponibiliza dois protocolos de transporte distintos para a camada de aplicação. Um deles é o **UDP** (User Datagram Protocol — Protocolo de Datagrama de Usuário), que oferece à aplicação solicitante um serviço não confiável, não orientado para conexão. O segundo é o **TCP** (Transmission Control Protocol — Protocolo de Controle de Transmissão), que oferece à aplicação solicitante um serviço confiável, orientado para conexão. Ao projetar uma aplicação de rede, o criador da aplicação deve especificar um desses dois protocolos de transporte. Como vimos nas Seções 2.7 e 2.8, o desenvolvedor da aplicação escolhe entre o UDP e o TCP ao criar *sockets*.

Para simplificar a terminologia, quando no contexto da Internet, faremos alusão ao pacote de camada de transporte como um *segmento*. Devemos mencionar, contudo, que a literatura da Internet (por exemplo, os RFCs) também se refere ao pacote de camada de transporte com/para TCP como um segmento, mas muitas vezes se refere ao pacote com/para UDP como um *datagrama*. Porém, a mesma literatura também usa o termo *datagrama* para o pacote de camada de rede! Como este é um livro de introdução a redes de computadores, acreditamos que será menos confuso se nos referirmos a ambos os pacotes TCP e UDP como segmentos; reservaremos o termo *datagrama* para o pacote de camada de rede.

Antes de continuarmos com nossa breve apresentação do UDP e do TCP, é útil dizer algumas palavras sobre a camada de rede da Internet. (A camada de rede é examinada detalhadamente no Capítulo 4.) O protocolo de camada de rede da Internet tem um nome — IP, que quer dizer *Internet Protocol*. O IP oferece comunicação lógica entre hospedeiros. O modelo de serviço do IP é um **serviço de entrega de melhor esforço**, o que significa que o IP faz o "melhor esforço" para levar segmentos entre hospedeiros comunicantes, *mas não dá nenhuma garantia*. Em especial, o IP não garante a entrega de segmentos, a entrega ordenada de segmentos e tampouco a integridade dos dados nos segmentos. Por essas razões, ele é denominado um **serviço não confiável**. Mencionamos também neste livro que cada hospedeiro tem, no mínimo, um endereço de camada de rede, denominado endereço IP. Examinaremos endereçamento IP em detalhes no Capítulo 4. Para este capítulo, precisamos apenas ter em mente que *cada hospedeiro tem um endereço IP*.

Agora que abordamos de modo breve o modelo de serviço IP, vamos resumir os modelos de serviço providos por UDP e TCP. A responsabilidade fundamental do UDP e do TCP é ampliar o serviço de entrega IP entre dois sistemas finais para um serviço de entrega entre dois processos que rodam nos sistemas finais. A ampliação da entrega hospedeiro a hospedeiro para entrega processo a processo é denominada **multiplexação/demultiplexação de camada de transporte**. Discutiremos esse assunto na próxima seção. O UDP e o TCP também fornecem verificação de integridade ao incluir campos de detecção de erros nos cabeçalhos de seus segmentos. Esses dois

serviços mínimos de camada de transporte — entrega de dados processo a processo e verificação de erros — são os únicos que o UDP fornece! Em especial, como o IP, o UDP é um serviço não confiável — ele não garante que os dados enviados por um processo cheguem (quando chegam!) intactos ao processo destinatário. O UDP será discutido detalhadamente na Seção 3.3.

O TCP, por outro lado, oferece vários serviços adicionais às aplicações. Primeiro, e mais importante, ele oferece **transferência confiável de dados**. Usando controle de fluxo, números de sequência, reconhecimentos e temporizadores (técnicas que exploraremos em pormenores neste capítulo), o protocolo assegura que os dados sejam entregues do processo remetente ao processo destinatário corretamente e em ordem. Assim, o TCP converte o serviço não confiável do IP entre sistemas finais em um serviço confiável de transporte de dados entre processos. Ele também oferece **controle de congestionamento**, que não é tanto um serviço fornecido à aplicação solicitante, e sim mais um serviço dirigido à Internet como um todo — para o bem geral. Em termos genéricos, o controle de congestionamento do TCP evita que qualquer outra conexão TCP abarrote os enlaces e roteadores entre hospedeiros comunicantes com uma quantidade excessiva de tráfego. Em princípio, o TCP permite que conexões TCP trafegando por um enlace de rede congestionado compartilhem em pé de igualdade a largura de banda daquele enlace. Isso é feito pela regulagem da taxa com a qual o lado remetente do TCP pode enviar tráfego para a rede. O tráfego UDP, por outro lado, não é regulado. Uma aplicação que usa transporte UDP pode enviar tráfego à taxa que quiser, pelo tempo que quiser.

Um protocolo que fornece transferência confiável de dados e controle de congestionamento é, necessariamente, complexo. Precisaremos de várias seções para detalhar os princípios da transferência confiável de dados e do controle de congestionamento, bem como de seções adicionais para explicar o protocolo TCP. Esses tópicos são analisados nas Seções 3.4 a 3.8. A abordagem escolhida neste capítulo é alternar entre princípios básicos e o protocolo TCP. Por exemplo, discutiremos primeiro a transferência confiável de dados em âmbito geral e, em seguida, como o TCP fornece especificamente a transferência confiável de dados. De maneira semelhante, iniciaremos discutindo o controle de congestionamento em âmbito geral e, em seguida, como o TCP realiza o controle de congestionamento. Porém, antes de chegarmos a essa parte boa, vamos examinar, primeiro, multiplexação/demultiplexação na camada de transporte.

3.2 MULTIPLEXAÇÃO E DEMULTIPLEXAÇÃO

Nesta seção, discutiremos multiplexação e demultiplexação na camada de transporte, isto é, a ampliação do serviço de entrega hospedeiro a hospedeiro provido pela camada de rede para um serviço de entrega processo a processo para aplicações que rodam nesses hospedeiros. Para manter a discussão em nível concreto, vamos examinar esse serviço básico da camada de transporte no contexto da Internet. Enfatizamos, contudo, que o serviço de multiplexação/demultiplexação é necessário para todas as redes de computadores.

No hospedeiro de destino, a camada de transporte recebe segmentos da camada de rede logo abaixo dela e tem a responsabilidade de entregar os dados desses segmentos ao processo de aplicação apropriado que roda no hospedeiro. Vamos examinar um exemplo. Suponha que você esteja sentado à frente de seu computador, baixando páginas Web enquanto roda uma sessão FTP e duas sessões Telnet. Por conseguinte, você tem quatro processos de aplicação de rede em execução — dois Telnet, um FTP e um HTTP. Quando a camada de transporte em seu computador receber dados da camada de rede abaixo dela, precisará direcionar os dados recebidos a um desses quatro processos. Vamos ver agora como isso é feito.

Em primeiro lugar, lembre-se de que dissemos, na Seção 2.7, que um processo (como parte de uma aplicação de rede) pode ter um ou mais ***sockets***, portas pelas quais dados passam da rede para o processo e do processo para a rede. Assim, como mostra a Figura 3.2, a camada de transporte do hospedeiro destinatário na verdade não entrega dados diretamente a um processo, mas a um *socket* intermediário. Como, a qualquer dado instante, pode haver mais de um *socket* no hospedeiro destinatário, cada um tem um identificador exclusivo. O formato do identificador depende de o *socket* ser UDP ou TCP, como discutiremos em breve.

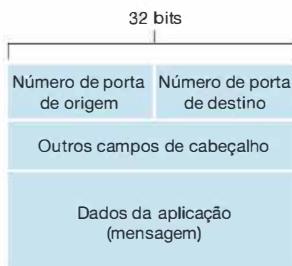



FIGURA 3.2 MULTIPLEXAÇÃO E DEMULTIPLEXAÇÃO NA CAMADA DE TRANSPORTE

Agora, vamos considerar como um hospedeiro destinatário direciona, ao *socket* apropriado, um segmento de camada de transporte que chega. Cada segmento de camada de transporte tem um conjunto de campos para tal finalidade. Na extremidade receptora, a camada de transporte examina esses campos para identificar a porta receptora e direcionar o segmento a esse *socket*. A tarefa de entregar os dados contidos em um segmento da camada de transporte ao *socket* correto é denominada **demultiplexação**. O trabalho de reunir, no hospedeiro de origem, partes de dados provenientes de diferentes *sockets*, encapsular cada parte de dados com informações de cabeçalho (que mais tarde serão usadas na demultiplexação) para criar segmentos, e passar esses segmentos para a camada de rede é denominada **multiplexação**. Note que a camada de transporte do hospedeiro que está no meio da Figura 3.2 tem de demultiplexar segmentos que chegam da camada de rede abaixo para os processos P_1 ou P_2 acima; isso é feito direcionando, ao *socket* do processo correspondente, os dados contidos no segmento que está chegando. A camada de transporte desse hospedeiro do meio também tem de juntar dados de saída desses *sockets*, formar segmentos de camada de transporte e passá-los à camada de rede. Embora tenhamos apresentado multiplexação e demultiplexação no contexto dos protocolos de transporte da Internet, é importante entender que essas operações estarão presentes sempre que um único protocolo em uma camada (na de transporte ou em qualquer outra) for usado por vários protocolos na camada mais alta seguinte.

Para ilustrar o serviço de demultiplexação, lembre-se da metáfora das moradias apresentada na seção anterior. Cada criança é identificada por seu nome próprio. Quando Bill recebe um lote de correspondência do carteiro, realiza uma operação de demultiplexação ao examinar a quem as cartas estão endereçadas e, em seguida, entregar a correspondência a seus irmãos e irmãs. Ann realiza uma operação de multiplexação quando coleta as cartas de seus irmãos e irmãs e entrega a correspondência na agência do correio.

Agora que entendemos os papéis da multiplexação e da demultiplexação na camada de transporte, vamos examinar como isso é feito em um hospedeiro. Sabemos, pela discussão anterior, que multiplexação na camada de rede requer (1) que as portas tenham identificadores exclusivos e (2) que cada segmento tenha campos especiais que indiquem a porta para a qual o segmento deve ser entregue. Esses campos especiais, ilustrados na Figura 3.3, são o **campo de número de porta de origem** e o campo de **número de porta de destino**. (Os segmentos UDP e TCP têm outros campos também, que serão examinados nas seções subsequentes deste capítulo.) Cada número de porta é um número de 16 bits na faixa de 0 a 65535. Os números de porta entre 0 e 1023 são denominados **números de porta bem conhecidos**; eles são restritos, o que significa que estão reservados para utilização por protocolos de aplicação bem conhecidos, como HTTP (que usa a porta número 80) e FTP (que usa a porta número 21). A lista dos números de porta bem conhecidos é apresentada no RFC 1700 e atualizada em <http://www.iana.org> [RFC 3232]. Quando desenvolvemos uma nova aplicação (como as desenvolvidas na Seção 2.7), devemos atribuir a ela um número de porta.

FIGURA 3.3 CAMPOS DE NÚMERO DE PORTA DE ORIGEM E DE DESTINO EM UM SEGMENTO DE CAMADA DE TRANSPORTE

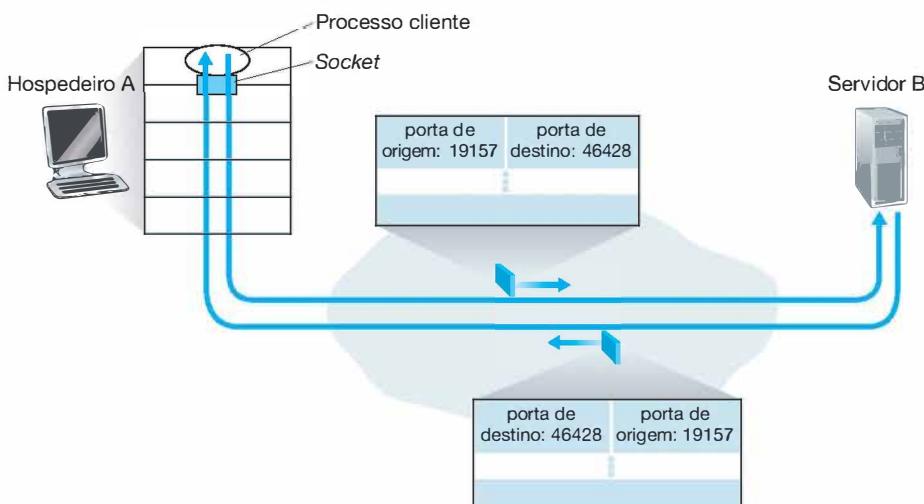
Agora já deve estar claro como a camada de transporte *poderia* realizar o serviço de demultiplexação: cada *socket* do hospedeiro pode receber um número designado; quando um segmento chega ao hospedeiro, a camada de transporte examina seu número de porta de destino e direciona o segmento ao *socket* correspondente. Então, os dados do segmento passam pela porta e entram no processo ligado a ela. Como veremos, é assim que o UDP faz demultiplexação. Todavia, veremos também que multiplexação/demultiplexação em TCP é ainda mais sutil.

Multiplexação e demultiplexação não orientadas para conexão

Lembre-se, na Seção 2.7.1, de que um programa em Python que roda em um hospedeiro pode criar uma porta UDP com a linha

```
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
```

Quando um *socket* UDP é criado dessa maneira, a camada de transporte automaticamente designa um número de porta ao *socket*. Em especial, a camada de transporte designa um número de porta na faixa de 1024 a 65535 que não esteja sendo usado naquele instante por qualquer outro *socket* UDP no hospedeiro. Como alternativa, podemos incluir uma linha em nosso programa Python depois de criarmos o *socket* para associar um número de porta específico (digamos, 19157) a este *socket* UDP por meio do método `bind()` do *socket*:

```
clientSocket.bind(('', 19157))
```

Se o desenvolvedor responsável por escrever o código da aplicação estivesse executando o lado servidor de um "protocolo bem conhecido", ele teria de designar o número de porta bem conhecido correspondente. O lado cliente da aplicação em geral permite que a camada de transporte designe o número de porta de modo automático (e transparente), ao passo que o lado servidor da aplicação designa um número de porta específico.

Agora que os *sockets* UDP já têm seus números de porta designados, podemos descrever multiplexação/demultiplexação UDP com precisão. Suponha que um processo no hospedeiro A, cujo número de porta UDP é 19157, queira enviar uma porção de dados de aplicação a um processo cujo número de porta UDP seja 46428 no hospedeiro B. A camada de transporte no hospedeiro A cria um segmento de camada de transporte que inclui os dados de aplicação, o número da porta de origem (19157), o número da porta de destino (46428) e mais outros dois valores (que serão discutidos mais adiante, mas que não são importantes para a discussão em curso). Então, a camada de transporte passa o segmento resultante para a camada de rede. Essa camada encapsula o segmento em um datagrama IP e faz uma tentativa de melhor esforço para entregar o segmento ao hospedeiro destinatário. Se o segmento chegar à máquina de destino B, a camada de destino no hospedeiro destinatário examinará o número da porta de destino no segmento (46428) e o entregará a seu *socket* identificado pelo número 46428. Note que a máquina B poderia estar rodando vários processos, cada um com sua própria porta UDP e número de porta associado. À medida que segmentos UDP chegassem da rede, a máquina B direcionaria (demultiplexaria) cada segmento à porta apropriada examinando o número de porta de destino do segmento.

É importante notar que um *socket* UDP é totalmente identificado por uma tupla com dois elementos, consistindo em um endereço IP de destino e um número de porta de destino. Por conseguinte, se dois segmentos UDP tiverem endereços IP de origem e/ou números de porta de origem diferentes, porém o mesmo endereço IP de *destino* e o mesmo número de porta de *destino*, eles serão direcionados ao mesmo processo de destino por meio do mesmo *socket* de destino.

É possível que agora você esteja imaginando qual é a finalidade do número da porta de origem. Como mostra a Figura 3.4, no segmento A-B, o número da porta de origem serve como parte de um "endereço de retorno" — quando B quer enviar um segmento de volta para A, a porta de destino no segmento B-A tomará seu valor do valor da porta de origem do segmento A-B. (O endereço de retorno completo é o endereço IP e o número de porta de origem de A.) Como exemplo, lembre-se do programa servidor UDP que estudamos na Seção 2.7. Em `UDPServer.py`, o servidor usa um método `recvfrom()` para extrair o número de porta cliente-servidor (de origem) do segmento que recebe do cliente; então envia um novo segmento ao cliente, com o número de porta que extraiu servindo como o número de porta de destino desse novo segmento.

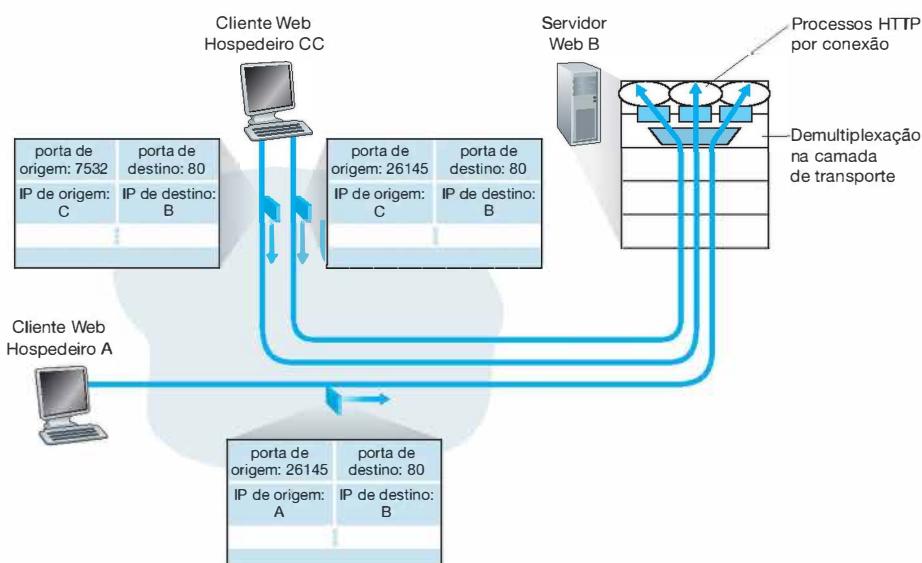
FIGURA 3.4 INVERSÃO DOS NÚMEROS DE PORTA DE ORIGEM E DESTINO

Multiplexação e demultiplexação orientadas para conexão

Para entender demultiplexação TCP, temos de examinar de perto *sockets* TCP e estabelecimento de conexão TCP. Há uma diferença sutil entre um *socket* UDP e um TCP: o *socket* TCP é identificado por uma tupla de quatro elementos: (endereço IP de origem, número da porta de origem, endereço IP de destino, número da porta de destino). Assim, quando um segmento TCP que vem da rede chega a um hospedeiro, este usa todos os quatro valores para direcionar (demultiplexar) o segmento para o *socket* apropriado. Em especial, e ao contrário do UDP, dois segmentos TCP chegando com endereços IP de origem ou números de porta de origem diferentes serão direcionados para dois *sockets* diferentes (com exceção de um TCP que esteja carregando a requisição de estabelecimento de conexão original). Para perceber melhor, vamos considerar novamente o exemplo de programação cliente-servidor TCP apresentado na Seção 2.7.2:

- A aplicação servidor TCP tem um "*socket* de entrada" que espera requisições de estabelecimento de conexão vindas de clientes TCP (ver Figura 2.29) na porta número 12000.
- O cliente TCP cria um *socket* e envia um segmento de requisição de estabelecimento de conexão com as linhas:

```
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,12000))
```

- Uma requisição de estabelecimento de conexão nada mais é do que um segmento TCP com número de porta de destino 12000 e um bit especial de estabelecimento de conexão marcado no cabeçalho TCP (que será tratado na Seção 3.5). O segmento inclui também um número de porta de origem que foi escolhido pelo cliente.
- Quando o sistema operacional do computador que está rodando o processo servidor recebe o segmento de requisição de conexão que está chegando e cuja porta de destino é 12000, ele localiza o processo servidor que está à espera para aceitar uma conexão na porta número 12000. Então, o processo servidor cria um novo *socket*:

```
connectionSocket, addr = serverSocket.accept()
```

- A camada de transporte no servidor também nota os quatro valores seguintes no segmento de requisição de conexão: (1) o número da porta de origem no segmento, (2) o endereço IP do hospedeiro de origem, (3) o número da porta de destino no segmento e (4) seu próprio endereço IP. O *socket* de conexão recém-criado é identificado por esses quatro valores; todos os segmentos subsequentes que chegarem, cuja porta de origem, endereço IP de origem, porta de destino e endereço IP de destino combinar com esses quatro valores, serão demultiplexados para esse *socket*. Com a conexão TCP agora ativa, o cliente e o servidor podem enviar dados um para o outro.

O hospedeiro servidor pode suportar vários *sockets* TCP simultâneos, sendo cada qual ligado a um processo e identificado por sua própria tupla de quatro elementos. Quando um segmento TCP chega ao hospedeiro, todos os quatro campos (endereço IP da origem, porta de origem, endereço IP de destino, porta de destino) são usados para direcionar (demultiplexar) o segmento para o *socket* apropriado.

A situação é ilustrada na Figura 3.5, na qual o hospedeiro C inicia duas sessões HTTP com o servidor B, e o hospedeiro A inicia uma sessão HTTP com o servidor B. Os hospedeiros A e C e o servidor B possuem, cada um,

FIGURA 3.5 DOIS CLIENTES QUE USAM O MESMO NÚMERO DE PORTA DE DESTINO (80) PARA SE COMUNICAR COM A MESMA APLICAÇÃO DE SERVIDOR WEB

SEGURANÇA EM FOCO

Varredura de porta

Vimos que um processo servidor espera com paciência, em uma porta aberta, o contato de um cliente remoto. Algumas portas são reservadas para aplicações familiares (por exemplo, Web, FTP, DNS e os servidores SMTP); outras são utilizadas por convenção por aplicações populares (por exemplo, o Microsoft SQL Server 2000 ouve as solicitações na porta 1434 do UDP). Desta forma, se determinarmos que uma porta está aberta em um hospedeiro, talvez possamos mapeá-la para uma aplicação específica sendo executada no hospedeiro. Isso é muito útil para administradores de sistemas, que muitas vezes têm interesse em saber quais aplicações estão sendo executadas nos hospedeiros em suas redes. Porém, os atacantes, a fim de "examinarem o local", também querem saber quais portas estão abertas nos hospedeiros direcionados. Se o hospedeiro estiver sendo executado em uma aplicação com uma falha de segurança conhecida (por exemplo, um servidor SQL ouvindo em uma porta 1434 estava sujeito a estouro de buffer, permitindo que um usuário remoto execute um código arbitrário no hospedeiro vulnerável, uma falha explorada pelo *worm* Slammer [CERT, 2003-04]), então esse hospedeiro está pronto para o ataque.

Determinar quais aplicações estão ouvindo em quais portas é uma tarefa de certo modo fácil. De fato, há inúmeros programas de domínio público, denominados varredores de porta, que fazem exatamente isso. Talvez o mais utilizado seja o nmap, disponível gratuitamente em http://nmap.org e incluído na maioria das distribuições Linux. Para o TCP, o nmap varre portas sequencialmente, procurando por portas que aceitem conexões TCP. Para o UDP, o nmap de novo varre portas em sequência, procurando por portas UDP que respondam aos segmentos UDP transmitidos. Em ambos os casos, o nmap retorna uma lista de portas abertas, fechadas ou inalcançáveis. Um hospedeiro executando o nmap pode tentar varrer qualquer hospedeiro direcionado em *qualquer lugar* da Internet. Voltaremos a falar sobre o nmap na Seção 3.5.6, ao discutirmos gerenciamento da conexão TCP.

seu próprio endereço IP exclusivo: A, C e B, respectivamente. O hospedeiro C atribui dois números diferentes (26145 e 7532) da porta de origem às suas duas conexões HTTP. Como o hospedeiro A está escolhendo números de porta independentemente de C, ele poderia também atribuir um número da porta de origem 26145 à sua conexão HTTP. Apesar disso, o servidor B ainda será capaz de demultiplexar corretamente as duas conexões que têm o mesmo número de porta de origem, já que elas têm endereços IP de origem diferentes.

Servidores Web e TCP

Antes de encerrar esta discussão, é instrutivo falar um pouco mais sobre servidores Web e como eles usam números de porta. Considere um hospedeiro rodando um servidor Web, tal como um Apache, na porta 80. Quando clientes (por exemplo, navegadores) enviam segmentos ao servidor, *todos* os segmentos terão a porta de destino 80. Em especial, os segmentos que estabelecem a conexão inicial e os que carregam as mensagens de requisição HTTP, ambos terão a porta de destino 80. Como acabamos de descrever, o servidor distingue os segmentos dos diferentes clientes pelos endereços IP e pelos números da porta de origem.

A Figura 3.5 mostra um servidor Web que cria um novo processo para cada conexão. Como mostra a figura, cada um desses processos tem seu próprio *socket* de conexão pelo qual chegam requisições HTTP e são enviadas respostas HTTP. Mencionamos, contudo, que nem sempre existe uma correspondência unívoca entre *sockets* de conexão e processos. Na verdade, os servidores Web de alto desempenho atuais muitas vezes utilizam somente um processo, mas criam uma nova *thread* com um novo *socket* de conexão para cada nova conexão cliente. (Uma *thread* pode ser considerada um subprocesso leve.) Se você fez a primeira tarefa de programação do Capítulo 2, construiu um servidor Web que faz exatamente isso. Para um servidor desses, a qualquer dado instante pode haver muitos *sockets* de conexão (com identificadores diferentes) ligados ao mesmo processo.

Se o cliente e o servidor estiverem usando HTTP persistente, então, durante toda a conexão persistente, trocarão mensagens HTTP pelo mesmo *socket* do servidor. Todavia, se usarem HTTP não persistente, então uma nova conexão TCP é criada e encerrada para cada requisição/resposta e, portanto, um novo *socket* é criado e mais tarde encerrado para cada requisição/resposta. Essa criação e encerramento frequentes de *sockets* podem causar sério impacto sobre o desempenho de um servidor Web movimentado (embora o sistema operacional consiga usar várias estratégias para atenuar o problema). Aconselhamos o leitor interessado em questões de sistema operacional referentes a HTTP persistente e não persistente a consultar [Nielsen, 1997; Nahum, 2002].

Agora que já discutimos multiplexação e demultiplexação na camada de transporte, passemos à discussão de um dos protocolos da Internet, o UDP. Na próxima seção, veremos que o UDP acrescenta pouco mais ao protocolo da camada de rede do que um serviço de multiplexação/demultiplexação.

3.3 TRANSPORTE NÃO ORIENTADO PARA CONEXÃO: UDP

Nesta seção, examinaremos o UDP mais de perto, como ele funciona e o que ele faz. Aconselhamos o leitor a rever o material apresentado na Seção 2.1, que inclui uma visão geral do modelo de serviço UDP, e o da Seção 2.7.1, que discute a programação de portas por UDP.

Para motivar nossa discussão sobre UDP, suponha que você esteja interessado em projetar um protocolo de transporte simples, bem básico. Como faria isso? De início, você deve considerar a utilização de um protocolo de transporte vazio. Em especial, do lado do remetente, considere pegar as mensagens do processo da aplicação e passá-las diretamente para a camada de rede; do lado do destinatário, considere pegar as mensagens que chegam da camada de rede e passá-las diretamente ao processo da aplicação. Mas, como aprendemos na seção anterior, o que teremos de fazer é praticamente nada. No mínimo, a camada de transporte tem de fornecer um serviço de multiplexação/demultiplexação para passar os dados da camada de rede ao processo em nível de aplicação correto.

O UDP, definido no [RFC 768], faz apenas quase tão pouco quanto um protocolo de transporte pode fazer. À parte sua função de multiplexação/demultiplexação e de alguma verificação de erros simples, ele nada adiciona ao IP. Na verdade, se o desenvolvedor de aplicação escolher o UDP, em vez do TCP, a aplicação estará "falando" quase diretamente com o IP. O UDP pega as mensagens do processo da aplicação, anexa os campos de número da porta de origem e de destino para o serviço de multiplexação/demultiplexação, adiciona dois outros pequenos campos e passa o segmento resultante à camada de rede, que encapsula o segmento dentro de um datagrama IP e, em seguida, faz a melhor tentativa para entregar o segmento ao hospedeiro receptor. Se o segmento chegar ao hospedeiro receptor, o UDP usará o número de porta de destino para entregar os dados do segmento ao processo de aplicação correto. Note que, com o UDP, não há apresentação entre as entidades remetente e destinatária da camada de transporte antes de enviar um segmento. Por essa razão, dizemos que o UDP é *não orientado para conexão*.

O DNS é um exemplo de protocolo de camada de aplicação que usa o UDP. Quando a aplicação DNS em um hospedeiro quer fazer uma consulta, constrói uma mensagem de consulta DNS e passa a mensagem para o UDP. Sem realizar nenhuma apresentação com a entidade UDP que está funcionando no sistema final de destino, o UDP do lado do hospedeiro adiciona campos de cabeçalho à mensagem e passa o segmento resultante à camada de rede, que encapsula o segmento UDP em um datagrama e o envia a um servidor de nomes. A aplicação DNS no hospedeiro requisitante então espera por uma resposta à sua consulta. Se não receber uma resposta (possivelmente porque a rede subjacente perdeu a consulta ou a resposta), ela tentará enviar a consulta a outro servidor de nomes ou informará à aplicação consultante que não pode obter uma resposta.

É possível que agora você esteja imaginando por que um desenvolvedor de aplicação escolheria construir uma aplicação sobre UDP, em vez de sobre TCP. O TCP não é sempre preferível ao UDP, já que fornece serviço confiável de transferência de dados e o UDP não? A resposta é "não", pois muitas aplicações se adaptam melhor ao UDP pelas seguintes razões:

- *Melhor controle no nível da aplicação sobre quais dados são enviados e quando*. Com UDP, tão logo um processo de aplicação passe dados ao UDP, o protocolo os empacotará dentro de um segmento UDP e os passará imediatamente à camada de rede. O TCP, por outro lado, tem um mecanismo de controle de congestionamento que limita o remetente TCP da camada de transporte quando um ou mais enlaces entre os hospedeiros da origem e do destinatário ficam congestionados demais. O TCP também continuará a reenviar um segmento até que o hospedeiro destinatário reconheça a recepção desse segmento, pouco importando o tempo que a entrega confiável levar. Visto que aplicações de tempo real requerem uma taxa mínima de envio, não querem atrasar demais a transmissão de segmentos e podem tolerar alguma perda de dados, o modelo de serviço do TCP não é particularmente compatível com as necessidades dessas aplicações. Como discutiremos adiante, essas aplicações podem usar UDP e executar, como parte da aplicação, qualquer funcionalidade adicional necessária além do serviço de entrega de segmentos simples e básico do UDP.
- *Não há estabelecimento de conexão*. Como discutiremos adiante, o TCP usa uma apresentação de três vias antes de começar a transferir dados. O UDP simplesmente envia mensagens sem nenhuma preliminar formal e, assim, não introduz atraso algum para estabelecer uma conexão. Talvez seja esta a principal razão pela qual o DNS roda sobre UDP, e não sobre TCP — o DNS seria muito mais lento se rodasse em TCP. O HTTP usa o TCP, e não o UDP, porque a confiabilidade é fundamental para páginas Web com texto. Mas, como discutimos brevemente na Seção 2.2, o atraso de estabelecimento de uma conexão TCP é uma contribuição importante aos atrasos associados à recepção de documentos Web.
- *Não há estados de conexão*. O TCP mantém o estado de conexão nos sistemas finais. Esse estado inclui buffers de envio e recebimento, parâmetros de controle de congestionamento e parâmetros numéricos de sequência e de reconhecimento. Veremos na Seção 3.5 que essa informação de estado é necessária para implementar o serviço de transferência confiável de dados do TCP e para prover controle de congestionamento. O UDP, por sua vez, não mantém o estado de conexão e não monitora nenhum desses parâmetros. Por essa razão, um servidor dedicado a uma aplicação específica pode suportar um número muito maior de clientes ativos quando a aplicação roda sobre UDP e não sobre TCP.
- *Pequeno excesso de cabeçalho de pacote*. O segmento TCP tem 20 bytes de excesso (*overhead*) de cabeçalho, além dos dados para cada segmento, enquanto o UDP tem somente 8 bytes de excesso.

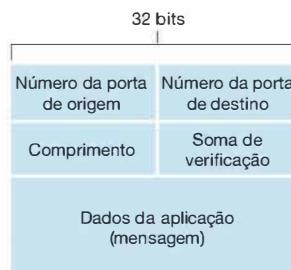
A Figura 3.6 relaciona aplicações populares da Internet e os protocolos de transporte que elas usam. Como era de esperar, o e-mail, o acesso a terminal remoto, a Web e a transferência de arquivos rodam sobre TCP — todas essas aplicações necessitam do serviço confiável de transferência de dados do TCP. Não obstante, muitas aplicações importantes executam sobre UDP, e não sobre TCP. O UDP é usado para atualização das tabelas de roteamento com o protocolo RIP (*Routing Information Protocol* — protocolo de informações de roteamento) (veja Seção 4.6.1). Como as atualizações RIP são enviadas periodicamente (em geral, a cada cinco minutos), atualizações perdidas serão substituídas por mais recentes, tornando inútil a recuperação das perdidas. O UDP também é usado para levar dados de gerenciamento de rede (SNMP; veja o Capítulo 9). Nesse caso, o UDP é preferível ao TCP, já que aplicações de gerenciamento de rede com frequência devem funcionar quando a rede está em estado sobrecarregado — exatamente quando é difícil conseguir transferência confiável de dados com congestionamento controlado. E também, como mencionamos, o DNS roda sobre UDP, evitando, desse modo, atrasos de estabelecimento de conexões TCP.

Como mostra a Figura 3.6, hoje o UDP e o TCP também são comumente usados para aplicações de multimídia, como telefone por Internet, videoconferência em tempo real e áudio e vídeo armazenados. Examinaremos essas aplicações mais de perto no Capítulo 7. No momento, mencionamos apenas que todas essas aplicações podem tolerar uma pequena quantidade de perda de pacotes, de modo que a transferência confiável de dados não é absolutamente crítica para o sucesso da aplicação. Além disso, aplicações em tempo real, como telefone por Internet e videoconferência, reagem muito mal ao controle de congestionamento do TCP. Por essas razões, os desenvolvedores de aplicações de multimídia muitas vezes optam por rodar suas aplicações sobre UDP em vez de sobre TCP. Entretanto, o TCP está sendo utilizado cada vez mais para transporte de mídia.

FIGURA 3.6 APLICAÇÕES POPULARES DA INTERNET E SEUS PROTOCOLOS DE TRANSPORTE SUBJACENTES

Aplicação	Protocolo da camada de aplicação	Protocolo de transporte subjacente
Correio eletrônico	SMTP	TCP
Acesso a terminal remoto	Telnet	TCP
Web	HTTP	TCP
Transferência de arquivo	FTP	TCP
Servidor de arquivo remoto	NFS	Tipicamente UDP
Recepção de multimídia	Tipicamente proprietário	UDP ou TCP
Telefonia por Internet	Tipicamente proprietário	UDP ou TCP
Gerenciamento de rede	SNMP	Tipicamente UDP
Protocolo de roteamento	RIP	Tipicamente UDP
Tradução de nome	DNS	Tipicamente UDP

Por exemplo, Sripanidkulchai [2004] descobriu que cerca de 75% do fluxo em tempo real e gravado utilizavam TCP. Quando as taxas de perda de pacote são baixas, junto com algumas empresas que bloqueiam o tráfego UDP por razões de segurança (veja Capítulo 8), o TCP se torna um protocolo cada vez mais atrativo para o transporte de mídia.

Embora hoje seja comum rodar aplicações de multimídia sobre UDP, isso é controvertido. Como já mencionamos, o UDP não tem controle de congestionamento. Mas esse controle é necessário para evitar que a rede entre em um estado no qual pouquíssimo trabalho útil é realizado. Se todos começassem a enviar vídeo com alta taxa de bits sem usar nenhum controle de congestionamento, haveria tamanho transbordamento de pacotes nos roteadores que poucos pacotes UDP conseguiriam atravessar com sucesso o caminho da origem ao destino. Além do mais, as altas taxas de perda induzidas pelos remetentes UDP sem controle fariam com que os remetentes TCP (que, como veremos adiante, *reduzem* suas taxas de envio em face de congestionamento) reduzissem drasticamente suas taxas. Assim, a falta de controle de congestionamento no UDP pode resultar em altas taxas de perda entre um remetente e um destinatário UDP e no acúmulo de sessões TCP — um problema potencialmente sério [Floyd, 1999]. Muitos pesquisadores propuseram novos mecanismos para forçar todas as origens, inclusive as origens UDP, a realizar um controle de congestionamento adaptativo [Mahdavi, 1997; Floyd, 2000; Kohler, 2006: RFC 4340].

Antes de discutirmos a estrutura do segmento UDP, mencionaremos que *é* possível uma aplicação ter transferência confiável de dados usando UDP. Isso pode ser feito se a confiabilidade for embutida na própria aplicação (por exemplo, adicionando mecanismos de reconhecimento e de retransmissão, tais como os que estudaremos na próxima seção). Mas essa é uma tarefa não trivial, que manteria o desenvolvedor ocupado com a depuração por um longo tempo. Não obstante, embutir confiabilidade diretamente na aplicação permite que ela tire proveito de ambas as alternativas. Em outras palavras, os processos da aplicação podem se comunicar de maneira confiável sem ter de se sujeitar às limitações de taxa de transmissão impostas pelo mecanismo de controle de congestionamento do TCP.

3.3.1 Estrutura do segmento UDP

A estrutura do segmento UDP, mostrada na Figura 3.7, é definida no RFC 768. Os dados da aplicação ocupam o campo de dados do segmento UDP. Por exemplo, para o DNS, o campo de dados contém uma mensagem de consulta ou uma mensagem de resposta. Para uma aplicação de recepção de áudio, amostras de áudio preenchem o campo de dados. O cabeçalho UDP tem apenas quatro campos, cada um consistindo em 2 bytes. Como já discutido na seção anterior, os números de porta permitem que o hospedeiro destinatário passe os dados da aplicação ao processo correto que está funcionando no sistema final destinatário (isto é, realize a função

de demultiplexação). O campo de comprimento especifica o número de bytes no segmento UDP (cabeçalho mais dados). Um valor de comprimento explícito é necessário porque o tamanho do campo de dados pode ser diferente de um segmento UDP para o outro. A soma de verificação é usada pelo hospedeiro receptor para verificar se foram introduzidos erros no segmento. Na verdade, a soma de verificação também é calculada para alguns dos campos no cabeçalho IP, além do segmento UDP. Mas ignoramos esse detalhe para podermos enxergar a floresta por entre as árvores. Discutiremos o cálculo da soma de verificação adiante. Os princípios básicos da detecção de erros estão descritos na Seção 5.2. O campo de comprimento especifica o comprimento do segmento UDP, incluindo o cabeçalho, em bytes.

FIGURA 3.7 ESTRUTURA DO SEGMENTO UDP

3.3.2 Soma de verificação UDP

A soma de verificação UDP serve para detectar erros. Em outras palavras, é usada para determinar se bits dentro do segmento UDP foram alterados (por exemplo, por ruído nos enlaces ou enquanto armazenados em um roteador) durante sua movimentação da origem até o destino. O UDP no lado remetente realiza o complemento de 1 da soma de todas as palavras de 16 bits do segmento levando em conta o "vai um" em toda a soma. Esse resultado é colocado no campo de soma de verificação no segmento UDP. Damos aqui um exemplo simples do cálculo da soma de verificação. Se quiser saber detalhes sobre a implementação eficiente do algoritmo de cálculo, consulte o RFC 1071; sobre o desempenho com dados reais, consulte Stone [1998 e 2000]. Como exemplo, suponha que tenhamos as seguintes três palavras de 16 bits:

0110011001100000
0101010101010101
1000111100001100

A soma das duas primeiras é:

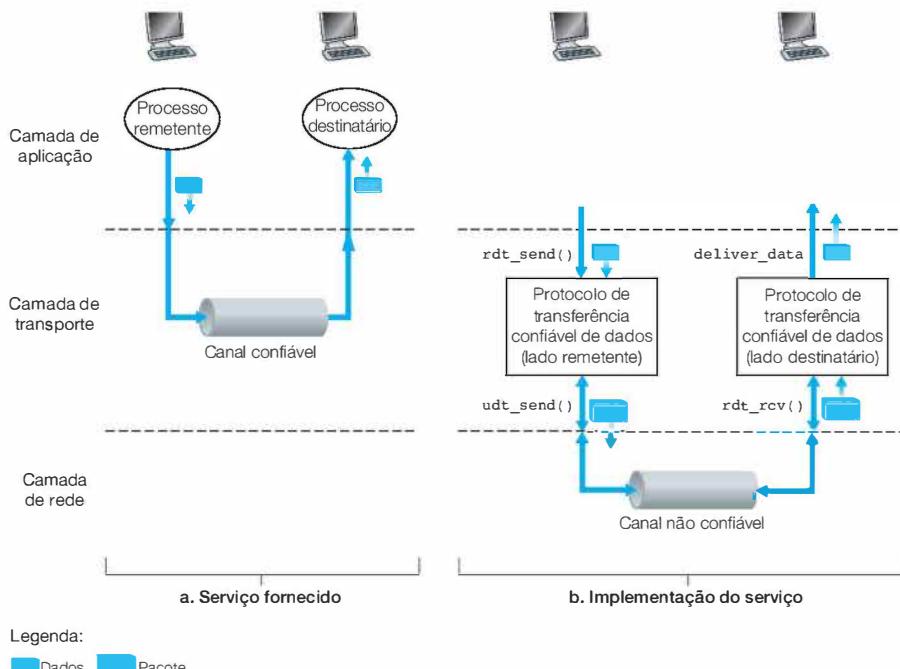
0110011001100000
0101010101010101
1011101110110101

Adicionando a terceira palavra à soma anterior, temos:

1011101110110101
1000111100001100
0100101011000010

Note que a última adição teve "vai um" no bit mais significativo que foi somado ao bit menos significativo. O complemento de 1 é obtido pela conversão de todos os 0 em 1 e de todos os 1 em 0. Desse modo, o complemento de 1 da soma 0100101011000010 é 1011010100111101, que passa a ser a soma de verificação. No destinatário, todas as quatro palavras de 16 bits são somadas, inclusive a soma de verificação. Se nenhum erro for introduzido no pacote, a soma no destinatário será, claro, 1111111111111111. Se um dos bits for um zero, saberemos então que um erro foi introduzido no pacote.

Talvez você esteja imaginando por que o UDP fornece uma soma de verificação primeiro, já que muitos protocolos de camada de enlace (dentre os quais, o popular Ethernet) também fornecem verificação de erros. A razão é que não há garantia de que todos os enlaces entre a origem e o destino forneçam tal verificação — um deles pode usar um protocolo de camada de enlace que não a forneça. Além disso, mesmo que os segmentos sejam corretamente transmitidos por um enlace, pode haver introdução de erros de bits quando um segmento é armazenado na memória de um roteador. Como não são garantidas nem a confiabilidade enlace a enlace, nem a detecção de erro na memória, o UDP deve prover detecção de erro *fim a fim* na camada de transporte se quisermos que o serviço de transferência de dados fim a fim forneça detecção de erro. É um exemplo do famoso **princípio fim a fim** do projeto de sistemas [Saltzer, 1984]. Tal princípio afirma que, visto ser dado como certo que funcionalidades (detecção de erro, neste caso) devem ser executadas fim a fim, "funções colocadas nos níveis mais baixos podem ser redundantes ou de pouco valor em comparação com o custo de fornecê-las no nível mais alto".

Como se pretende que o IP rode sobre qualquer protocolo de camada 2, é útil que a camada de transporte forneça verificação de erros como medida de segurança. Embora o UDP forneça verificação de erros, ele nada faz para recuperar-se de um erro. Algumas implementações do UDP apenas descartam o segmento danificado; outras passam o segmento errado à aplicação acompanhado de um aviso.

Isso encerra nossa discussão sobre o UDP. Logo veremos que o TCP oferece transferência confiável de dados a suas aplicações, bem como outros serviços que o UDP não oferece. Naturalmente, o TCP também é mais complexo do que o UDP. Contudo, antes de discutirmos o TCP, primeiro devemos examinar os princípios subjacentes da transferência confiável de dados.

3.4 PRINCÍPIOS DA TRANSFERÊNCIA CONFIÁVEL DE DADOS

Nesta seção, consideramos o problema conceitual da transferência confiável de dados. Isso é apropriado, já que o problema de realizar transferência confiável de dados ocorre não só na camada de transporte, mas também na de enlace e na de aplicação. Assim, o problema geral é de importância central para o trabalho em rede. Na verdade, se tivéssemos de fazer uma lista dos dez maiores problemas mais importantes para todo o trabalho em rede, o da transferência confiável de dados seria o candidato número um da lista. Na seção seguinte, examinaremos o TCP e mostraremos, em especial, que ele utiliza muitos dos princípios que descreveremos aqui.

A Figura 3.8 ilustra a estrutura para nosso estudo de transferência confiável de dados. A abstração do serviço fornecido às entidades das camadas superiores é a de um canal confiável através do qual dados podem ser transferidos. Com um canal confiável, nenhum dos dados transferidos é corrompido (trocado de 0 para 1 ou vice-versa) nem perdido, e todos são entregues na ordem em que foram enviados. Este é exatamente o modelo de serviço oferecido pelo TCP às aplicações de Internet que recorrem a ele.

É responsabilidade de um **protocolo de transferência confiável de dados** implementar essa abstração de serviço. A tarefa é dificultada pelo fato de que a camada *abaixo* do protocolo de transferência confiável de dados talvez seja não confiável. Por exemplo, o TCP é um protocolo confiável de transferência de dados que é executado sobre uma camada de rede fim a fim não confiável (IP). De modo mais geral, a camada abaixo das duas extremidades que se comunicam de modo confiável pode consistir em um único enlace físico (como no caso de um protocolo de transferência de dados na camada de enlace) ou em uma rede global interligada (como em um protocolo de camada de transporte). Para nossa finalidade, contudo, podemos considerar essa camada mais baixa apenas como um canal ponto a ponto não confiável.

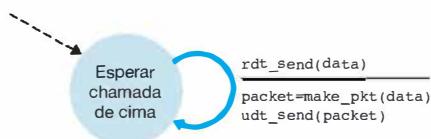
FIGURA 3.8 TRANSFERÊNCIA CONFIÁVEL DE DADOS: MODELO DO SERVIÇO E IMPLEMENTAÇÃO DO SERVIÇO

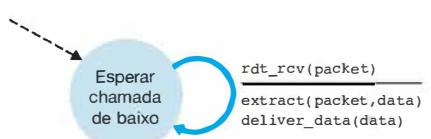
Nesta seção, desenvolveremos de modo gradual os lados remetente e destinatário de um protocolo confiável de transferência de dados, considerando modelos progressivamente mais complexos do canal subjacente. Por exemplo, vamos considerar que os mecanismos do protocolo são necessários quando o canal subjacente puder corromper bits ou perder pacotes inteiros. Uma suposição que adotaremos em toda essa discussão é que os pacotes serão entregues na ordem em que foram enviados, com alguns pacotes possivelmente sendo perdidos; ou seja, o canal subjacente não reordenará pacotes. A Figura 3.8(b) ilustra as interfaces de nosso protocolo de transferência de dados. O lado remetente do protocolo será invocado de cima, por uma chamada a `rdt_send()`. Ele passará os dados a serem entregues à camada superior no lado destinatário. (Aqui, `rdt` significa protocolo *reliable data transfer* — transferência confiável de dados — e `_send` indica que o lado remetente do rdt está sendo chamado. O primeiro passo no desenvolvimento de qualquer protocolo é dar-lhe um bom nome!) Do lado destinatário, `rdt_rcv()` será chamado quando um pacote chegar pelo lado destinatário do canal. Quando o protocolo `rdt` quiser entregar dados à camada superior, ele o fará chamando `deliver_data()`. No que se segue, usamos a terminologia "pacote" em vez de "segmento" de camada de transporte. Como a teoria desenvolvida nesta seção se aplica a redes de computadores em geral, e não só à camada de transporte da Internet, o termo genérico "pacote" talvez seja mais apropriado aqui.

Nesta seção, consideramos apenas o caso de **transferência unidirecional de dados**, isto é, transferência de dados do lado remetente ao lado destinatário. O caso de **transferência bidirecional confiável de dados** (isto é, *full-duplex*) não é conceitualmente mais difícil, mas é bem mais tedioso de explicar. Embora consideremos apenas a transferência unidirecional de dados, é importante notar que, apesar disso, os lados remetente e destinatário de nosso protocolo terão de transmitir pacotes em *ambas* as direções, como mostra a Figura 3.8. Logo veremos que, além de trocar pacotes contendo os dados a transferir, os lados remetente e destinatário do `rdt` também precisarão

trocar pacotes de controle entre si. Ambos os lados de envio e destino do `rdt` enviam pacotes para o outro por meio de uma chamada a `udt_send()` (em que udt significa *unreliable data transfer* — transferência não confiável de dados).

3.4.1 Construindo um protocolo de transferência confiável de dados

Vamos percorrer agora uma série de protocolos que vão se tornando cada vez mais complexos, até chegar a um protocolo de transferência confiável de dados impecável.

Transferência confiável de dados sobre um canal perfeitamente confiável: `rdt1.0`

Consideremos primeiro o caso mais simples, em que o canal subjacente é completamente confiável. O protocolo em si, que denominaremos `rdt1.0`, é trivial. As definições de **máquina de estado finito** (*finite-state machine* — **FSM**) para o remetente e o destinatário `rdt1.0` são apresentadas na Figura 3.9. A FSM da Figura 3.9(a) define a operação do remetente, enquanto a FSM da Figura 3.9(b) define a operação do destinatário. É importante notar que há FSM *separadas* para o remetente e o destinatário. Ambas as FSM da Figura 3.9 têm apenas um estado. As setas na descrição da FSM indicam a transição do protocolo de um estado para outro. (Como cada FSM da Figura 3.9 tem apenas um estado, uma transição é, necessariamente, de um dado estado para ele mesmo; examinaremos diagramas de estados mais complicados em breve.) O evento que causou a transição é mostrado acima da linha horizontal que a rotula, e as ações realizadas quando ocorre o evento são mostradas abaixo dessa linha. Quando nenhuma ação é realizada em um evento, ou quando não ocorre nenhum evento e uma ação é realizada, usaremos o símbolo Λ, acima ou abaixo da linha horizontal, para indicar a falta de uma ação ou de um evento, respectivamente. O estado inicial da FSM é indicado pela seta tracejada. Embora as FSM da Figura 3.9 tenham apenas um estado, as outras que veremos em breve têm vários, portanto, será importante identificar o estado inicial de cada FSM.

O lado remetente do rdt apenas aceita dados da camada superior pelo evento `rdt_send(data)`, cria um pacote que contém os dados (pela ação `make_pkt(data)`) e o envia para dentro do canal. Na prática, o evento `rdt_send(data)` resultaria de uma chamada de procedimento (por exemplo, para `rdt_send()`) pela aplicação da camada superior.

Do lado destinatário, rdt recebe um pacote do canal subjacente pelo evento `rdt_rcv(packet)`, extrai os dados do pacote (pela ação `extract(packet, data)`) e os passa para a camada superior (pela ação

FIGURA 3.9 `rdt1.0` **– UM PROTOCOLO PARA UM CANAL COMPLETAMENTE CONFIÁVEL**

deliver_data(data)). Na prática, o evento rdt_ rcv(packet) resultaria de uma chamada de procedimento (por exemplo, para rdt_rcv( )) do protocolo da camada inferior.

Nesse protocolo simples, não há diferença entre a unidade de dados e um pacote. E, também, todo o fluxo de pacotes corre do remetente para o destinatário; com um canal perfeitamente confiável, não há necessidade de o lado destinatário fornecer qualquer informação ao remetente, já que nada pode dar errado! Note que também admitimos que o destinatário está capacitado a receber dados seja qual for a velocidade em que o remetente os envie. Assim, não há necessidade de pedir para o remetente desacelerar!

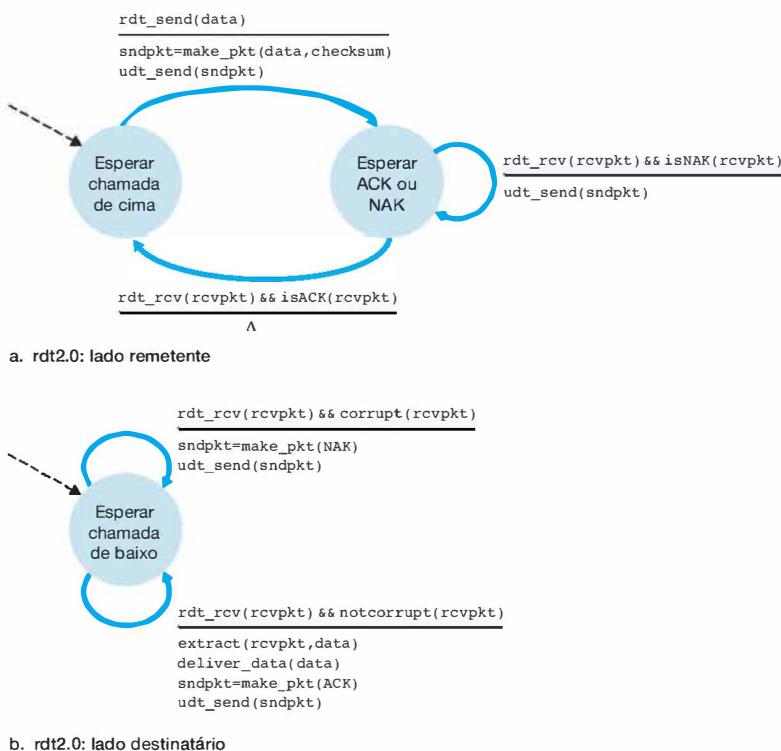
Transferência confiável de dados por um canal com erros de bits: rdt2.0

Um modelo mais realista de canal subjacente é um canal em que os bits de um pacote podem ser corrompidos. Esses erros de bits ocorrem em geral nos componentes físicos de uma rede enquanto o pacote é transmitido, propagado ou armazenado. Continuaremos a admitir, por enquanto, que todos os pacotes transmitidos sejam recebidos (embora seus bits possam estar corrompidos) na ordem em que foram enviados.

Antes de desenvolver um protocolo para se comunicar de maneira confiável com esse canal, considere primeiro como as pessoas enfrentariam uma situação como essa. Imagine como você ditaria uma mensagem longa pelo telefone. Em um cenário típico, quem estivesse anotando a mensagem diria "OK" após cada sentença que ouvisse, entendesse e anotasse. Se a pessoa ouvisse uma mensagem truncada, pediria que você a repetisse. Esse protocolo de ditado de mensagem usa **reconhecimentos positivos** ("OK") e **reconhecimentos negativos** ("Repita, por favor"). Tais mensagens de controle permitem que o destinatário faça o remetente saber o que foi recebido corretamente e o que foi recebido com erro e, portanto, exige repetição. Em um arranjo de rede de computadores, protocolos de transferência confiável de dados baseados nesse tipo de retransmissão são conhecidos como **protocolos ARQ (Automatic Repeat reQuest — solicitação automática de repetição)**.

Basicamente, são exigidas três capacitações adicionais dos protocolos ARQ para manipular a presença de erros de bits:

- *Detecção de erros.* Primeiro, é preciso um mecanismo que permita ao destinatário detectar quando ocorrem erros. Lembre-se de que dissemos na seção anterior que o UDP usa o campo de soma de verificação da Internet exatamente para essa finalidade. No Capítulo 5, examinaremos, com mais detalhes, técnicas de detecção e de correção de erros. Elas permitem que o destinatário detecte e talvez corrija erros de bits de pacotes. Por enquanto, basta saber que essas técnicas exigem que bits extras (além dos bits dos dados originais a serem transferidos) sejam enviados do remetente ao destinatário. Esses bits são colocados no campo de soma de verificação do pacote de dados do protocolo rdt2.0.
- *Realimentação do destinatário.* Como remetente e destinatário em geral estejam rodando em sistemas finais diferentes, possivelmente separados por milhares de quilômetros, o único modo de o remetente saber qual é a visão de mundo do destinatário (neste caso, se um pacote foi recebido corretamente ou não) é o destinatário fornecer realimentação explícita ao remetente. As respostas de reconhecimento positivo (ACK) ou negativo (NAK) no cenário do ditado da mensagem são exemplos dessa realimentação. Nosso protocolo rdt2.0 devolverá, dessa mesma maneira, pacotes ACK e NAK do destinatário ao remetente. Em princípio, esses pacotes precisam apenas ter o comprimento de um bit; por exemplo, um valor 0 poderia indicar um NAK e um valor 1 poderia indicar um ACK.
- *Retransmissão.* Um pacote que é recebido com erro no destinatário será retransmitido pelo remetente.

A Figura 3.10 mostra a representação por FSM do rdt2.0, um protocolo de transferência de dados que emprega detecção de erros, reconhecimentos positivos e reconhecimentos negativos.

O lado remetente do rdt2.0 tem dois estados. No estado mais à esquerda, o protocolo do lado remetente está esperando que os dados sejam passados pela camada superior. Quando o evento rdt_send(data) ocorrer, o remetente criará um pacote (sndpkt) contendo os dados a serem enviados, junto com uma soma de verificação do pacote (por exemplo, como discutimos na Seção 3.3.2 para o caso de um segmento UDP) e,

então, enviará o pacote pela operação `udt_send(sndpkt)`. No estado mais à direita, o protocolo remetente está esperando por um pacote ACK ou NAK da parte do destinatário. Se um pacote ACK for recebido (a notação `rdt_rcv(rcvpkt) && isACK (rcvpkt)` na Figura 3.10 corresponde a esse evento), o remetente saberá que o pacote transmitido mais recentemente foi recebido corretamente. Assim, o protocolo volta ao estado de espera por dados vindos da camada superior. Se for recebido um NAK, o protocolo retransmitirá o último pacote e esperará por um ACK ou NAK a ser devolvido pelo destinatário em resposta ao pacote de dados retransmitido. É importante notar que, quando o destinatário está no estado de espera por ACK ou NAK, *não pode* receber mais dados da camada superior; isto é, o evento `rdt_send()` não pode ocorrer; isso somente acontecerá após o remetente receber um ACK e sair desse estado. Assim, o remetente não enviará novos dados até ter certeza de que o destinatário recebeu corretamente o pacote em questão. Devido a esse comportamento, protocolos como o `rdt2.0` são conhecidos como protocolos **pare e espere** (*stop-and-wait*).

A FSM do lado destinatário para o `rdt2.0` tem um único estado. Quando o pacote chega, o destinatário responde com um ACK ou um NAK, dependendo de o pacote recebido estar ou não corrompido. Na Figura 3.10, a notação `rdt_rcv(rcvpkt) && corrupt(rcvpkt)` corresponde ao evento em que um pacote é recebido e existe um erro.

Pode parecer que o protocolo `rdt2.0` funciona, mas infelizmente ele tem um defeito fatal. Em especial, ainda não tratamos da possibilidade de o pacote ACK ou NAK estar corrompido! (Antes de continuar, é bom você começar a pensar em como esse problema pode ser resolvido.) Lamentavelmente, nossa pequena omissão não é tão inofensiva quanto possa parecer. No mínimo, precisaremos adicionar aos pacotes ACK/NAK bits de

FIGURA 3.10 `rdt2.0` **– UM PROTOCOLO PARA UM CANAL COM ERROS DE BITS**

rdt_send(data)
sndpkt=make_pkt(data,checksum)
udt_send(sndpkt)
Esperar chamada de cima
Esperar ACK ou NAK
rdt_rcv(rcvpkt) && isNAK(rcvpkt)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && isACK(rcvpkt)
Λ

a. rdt2.0: lado remetente

b. rdt2.0: lado destinatário

soma de verificação para detectar esses erros. A questão mais difícil é como o protocolo deve se recuperar de erros em pacotes ACK ou NAK. Nesse caso, a dificuldade é que, se um ACK ou um NAK estiver corrompido, o remetente não terá como saber se o destinatário recebeu ou não corretamente a última parte de dados transmitidos.

Considere três possibilidades para manipular ACKs ou NAKs corrompidos:

- Para a primeira possibilidade, imagine o que um ser humano faria no cenário do ditado da mensagem. Se quem estiver ditando não entender o "OK" ou o "Repita, por favor" do destinatário, provavelmente perguntará: "O que foi que você disse?" (introduzindo assim um novo tipo de pacote remetente-destinatário em nosso protocolo). O destinatário então repetiria a resposta. Mas e se a frase "O que foi que você disse?" estivesse corrompida? O destinatário, sem ter nenhuma noção se a sentença corrompida era parte do ditado ou um pedido para repetir a última resposta, provavelmente responderia: "O que foi que *você* disse?". E então, é claro, essa resposta também poderia estar truncada. É óbvio que estamos entrando em um caminho difícil.
- Uma segunda alternativa é adicionar um número suficiente de bits de soma de verificação para permitir que o remetente não só detecte, mas também se recupere de erros de bits. Isso resolve o problema imediato para um canal que pode corromper pacotes, mas não perdê-los.
- Uma terceira técnica é o remetente reenviar o pacote de dados corrente quando receber um pacote ACK ou NAK truncado. Esse método, no entanto, introduz **pacotes duplicados** no canal remetente-destinatário. A dificuldade fundamental com pacotes duplicados é que o destinatário não sabe se o último ACK ou NAK que enviou foi bem recebido no remetente. Assim, ele não pode saber *a priori* se um pacote que chega contém novos dados ou se é uma retransmissão!

Uma solução simples para esse novo problema (e que é adotada em quase todos os protocolos de transferência de dados existentes, inclusive o TCP) é adicionar um novo campo ao pacote de dados e fazer o remetente numerar seus pacotes de dados colocando um **número de sequência** nesse campo. O destinatário então teria apenas de verificar esse número de sequência para determinar se o pacote recebido é ou não uma retransmissão. Para esse caso simples de protocolo pare e espere, um número de sequência de um bit é suficiente, já que permitirá que o destinatário saiba se o remetente está reenviando o pacote previamente transmitido (o número de sequência do pacote recebido é o mesmo do pacote recebido mais recentemente) ou um novo pacote (o número de sequência muda, indo "para a frente" em progressão aritmética de módulo 2). Como estamos admitindo que este é um canal que não perde pacotes, os pacotes ACK e NAK em si não precisam indicar o número de sequência do pacote que estão reconhecendo. O remetente sabe que um pacote ACK ou NAK recebido (truncado ou não) foi gerado em resposta ao seu pacote de dados transmitidos mais recentemente.

As figuras 3.11 e 3.12 mostram a descrição da FSM para o `rdt2.1`, nossa versão corrigida do `rdt2.0`. Cada `rdt2.1` remetente e destinatário da FSM agora tem um número duas vezes maior de estados do que antes. Isso acontece porque o estado do protocolo deve agora refletir se o pacote que está sendo correntemente enviado (pelo remetente) ou aguardado (no destinatário) deveria ter um número de sequência 0 ou 1. Note que as ações nos estados em que um pacote numerado com 0 está sendo enviado ou aguardado são imagens especulares daquelas que devem funcionar quando estiver sendo enviado ou aguardado um pacote numerado com 1; as únicas diferenças têm a ver com a manipulação do número de sequência.

O protocolo `rdt2.1` usa tanto o reconhecimento positivo como o negativo do remetente ao destinatário. Quando um pacote fora de ordem é recebido, o destinatário envia um reconhecimento positivo para o pacote que recebeu; quando um pacote corrompido é recebido, ele envia um reconhecimento negativo. Podemos conseguir o mesmo efeito de um pacote NAK se, em vez de enviarmos um NAK, enviarmos um ACK em seu lugar para o último pacote corretamente recebido. Um remetente que recebe dois ACKs para o mesmo pacote (isto é, **ACKs duplicados**) sabe que o destinatário não recebeu corretamente o pacote seguinte àquele para o qual estão sendo dados dois ACKs. Nosso protocolo de transferência confiável de dados sem NAK para um canal com erros de bits é o `rdt2.2`, mostrado nas figuras 3.13 e 3.14. Uma modificação sutil entre `rdt2.1` e `rdt2.2` é que o destinatário agora deve incluir o número de sequência do pacote que está sendo reconhecido por uma mensagem ACK

(o que é feito incluindo o argumento ACK, 0 ou ACK, 1 em `make_pkt()` na FSM destinatária) e o remetente agora deve verificar o número de sequência do pacote que está sendo reconhecido por uma mensagem ACK recebida (o que é feito incluindo o argumento 0 ou 1 em `isACK()` na FSM remetente).

FIGURA 3.11 `rdt2.1` **REMETENTE**

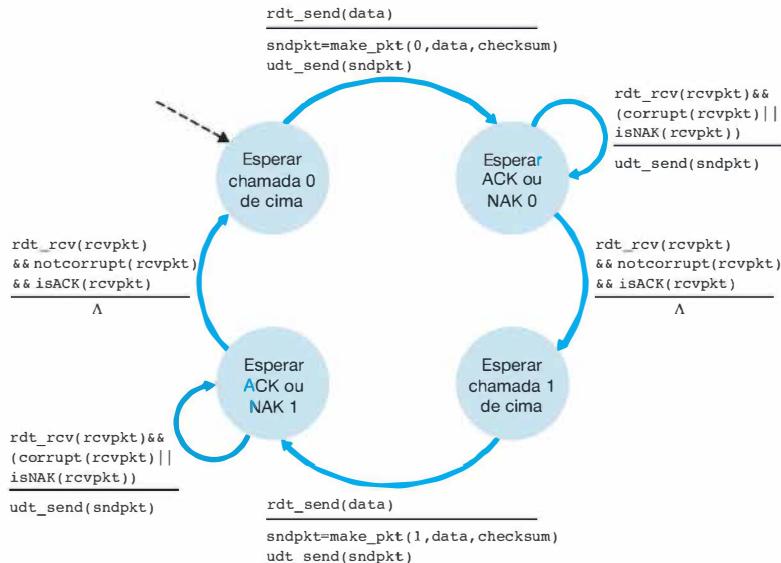



FIGURA 3.12 `rdt2.1` **DESTINATÁRIO**

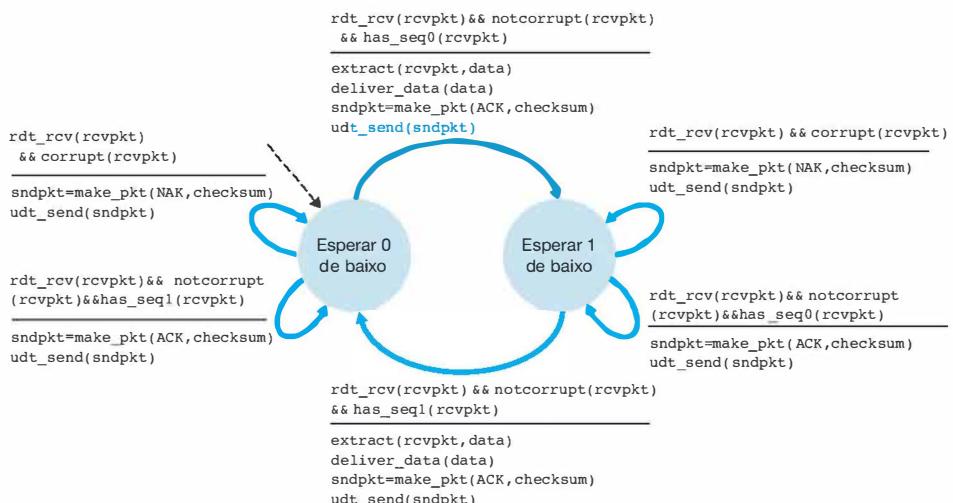
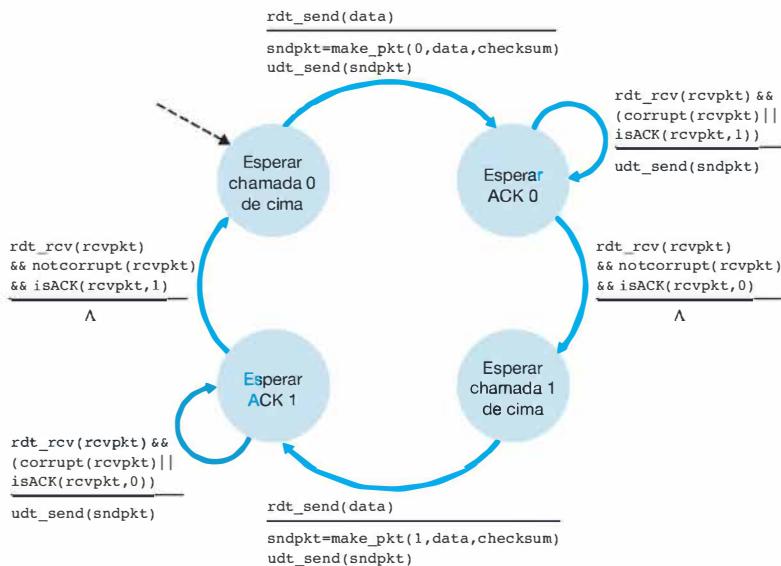
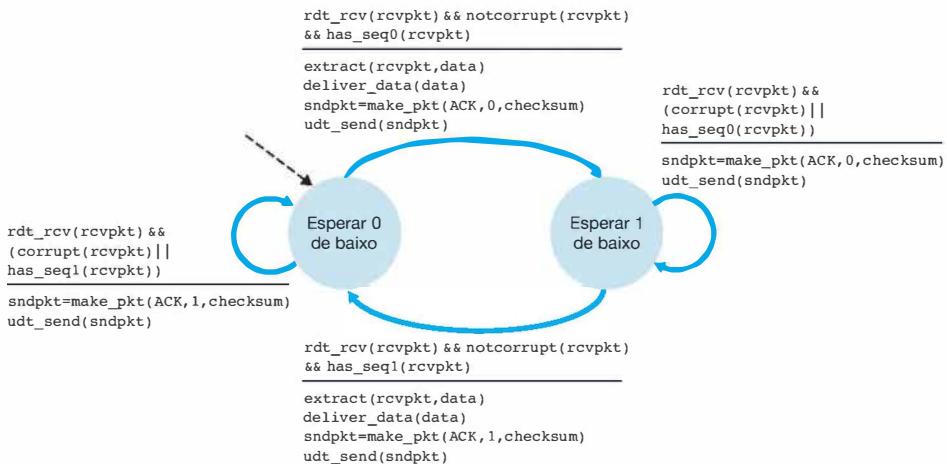
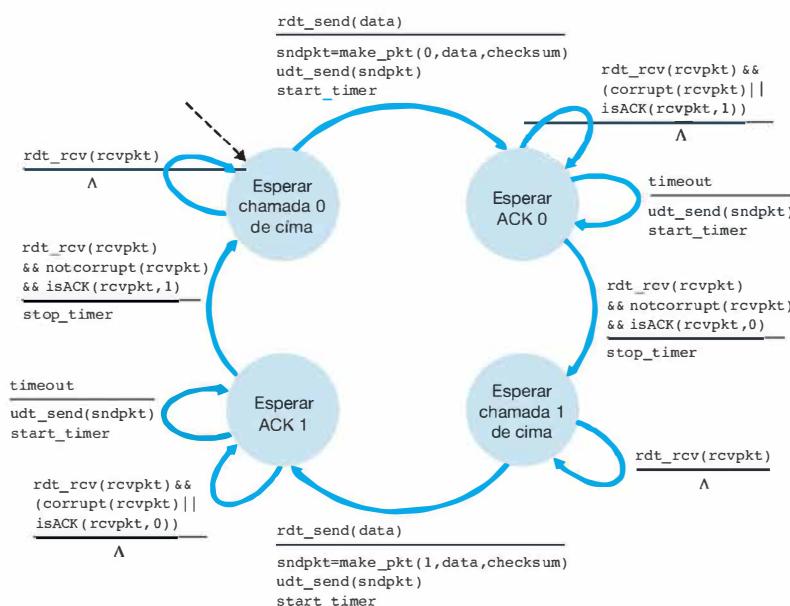



FIGURA 3.13 `rdt2.2` **REMETENTE**

FIGURA 3.14 `rdt2.2` **DESTINATÁRIO**

Transferência confiável de dados por um canal com perda e com erros de bits: `rdt3.0`

Suponha agora que, além de corromper bits, o canal subjacente possa perder pacotes, um acontecimento que não é incomum nas redes de computadores de hoje (incluindo a Internet). Duas preocupações adicionais devem agora ser tratadas pelo protocolo: como detectar perda de pacote e o que fazer quando isso ocorre. A utilização de soma de verificação, números de sequência, pacotes ACK e retransmissões — as técnicas já desenvolvidas em `rdt2.2` — nos permitirá atender a última preocupação. Lidar com a primeira preocupação, por sua vez, exigirá a adição de um novo mecanismo de protocolo.

Há muitas abordagens possíveis para lidar com a perda de pacote (e diversas delas serão estudadas nos exercícios ao final do capítulo). Aqui, atribuiremos ao remetente o encargo de detectar e se recuperar das perdas de pacote. Suponha que o remetente transmita um pacote de dados e que esse pacote, ou o ACK do seu destinatário, seja perdido. Em qualquer um dos casos, nenhuma resposta chegará ao remetente vinda do destinatário. Se o remetente estiver disposto a esperar o tempo suficiente para ter *certeza* de que o pacote foi perdido, ele poderá apenas retransmitir o pacote de dados. É preciso que você se convença de que esse protocolo funciona mesmo.

Mas quanto o remetente precisa esperar para ter certeza de que algo foi perdido? É claro que deve aguardar no mínimo o tempo de um atraso de ida e volta entre ele e o destinatário (o que pode incluir buffers em roteadores ou equipamentos intermediários) e mais o tempo que for necessário para processar um pacote no destinatário. Em muitas redes, o atraso máximo para esses piores casos é muito difícil até de estimar, quanto mais saber com certeza. Além disso, o ideal seria que o protocolo se recuperasse da perda de pacotes logo que possível; esperar pelo atraso do pior dos casos pode significar um longo tempo até que a recuperação do erro seja iniciada. Assim, a técnica adotada na prática é a seguinte: o remetente faz uma escolha ponderada de um valor de tempo dentro do qual seria provável, mas não garantido, que a perda tivesse acontecido. Se não for recebido um ACK nesse período, o pacote é retransmitido. Note que, se um pacote sofrer um atraso particularmente longo, o remetente poderá retransmiti-lo mesmo que nem o pacote de dados, nem o seu ACK tenham sido perdidos. Isso introduz a possibilidade de **pacotes de dados duplicados** no canal remetente-destinatário. Felizmente, o protocolo `rdt2.2` já dispõe de funcionalidade suficiente (isto é, números de sequência) para tratar dos casos de pacotes duplicados.

Do ponto de vista do remetente, a retransmissão é uma panaceia. O remetente não sabe se um pacote de dados foi perdido, se um ACK foi perdido ou se o pacote ou o ACK apenas estavam muito atrasados. Em todos os casos, a ação é a mesma: retransmitir. Para implementar um mecanismo de retransmissão com base no tempo, é necessário um **temporizador de contagem regressiva** que interrompa o processo remetente após ter decorrido um dado tempo. Assim, será preciso que o remetente possa (1) acionar o temporizador todas as vezes que um pacote for enviado (quer seja a primeira vez, quer seja uma retransmissão), (2) responder a uma interrupção feita pelo temporizador (realizando as ações necessárias) e (3) parar o temporizador.

A Figura 3.15 mostra a FSM remetente para o `rdt3.0`, um protocolo que transfere dados de modo confiável por um canal que pode corromper ou perder pacotes; nos "Exercícios de fixação" pediremos a você que projete a

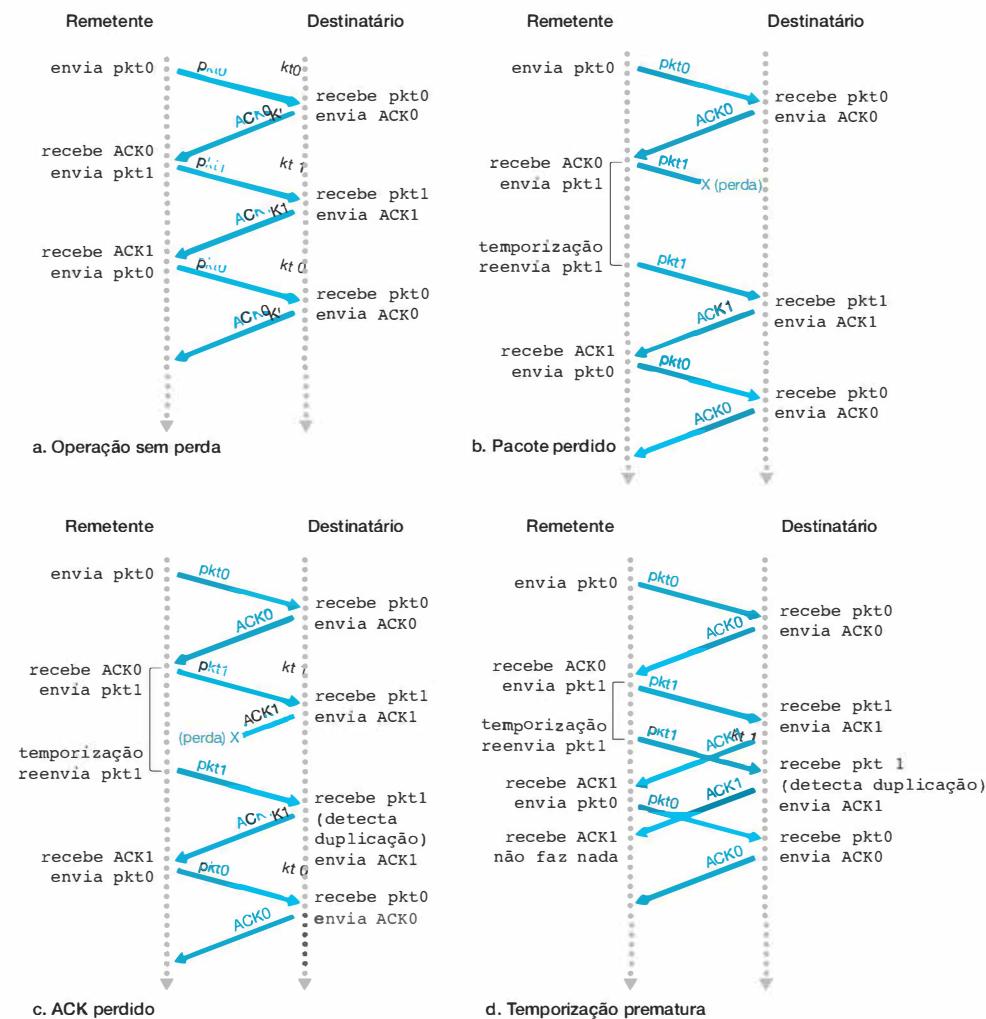



FIGURA 3.15 **`rdt3.0` REMETENTE**

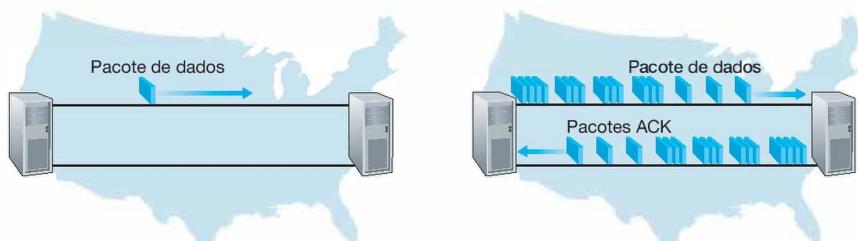
FSM destinatária para `rdt3.0`. A Figura 3.16 mostra como o protocolo funciona sem pacotes perdidos ou atrasados e como manipula pacotes de dados perdidos. Nessa figura, a passagem do tempo ocorre do topo do diagrama para baixo. Note que o instante de recebimento de um pacote tem de ser posterior ao instante de envio de um pacote, como resultado de atrasos de transmissão e de propagação. Nas figuras 3.16(b-d), os colchetes do lado remetente indicam os instantes em que o temporizador foi acionado e, mais tarde, os instantes em que ele parou. Vários dos aspectos mais sutis desse protocolo são examinados nos exercícios ao final deste capítulo. Como os números de sequência se alternam entre 0 e 1, o protocolo `rdt3.0` às vezes é conhecido como **protocolo bit alternante**.

Agora já reunimos os elementos fundamentais de um protocolo de transferência de dados. Somas de verificação, números de sequência, temporizadores e pacotes de reconhecimento negativo e positivo — cada um desempenha um papel crucial e necessário na operação do protocolo. Temos agora em funcionamento um protocolo de transferência confiável de dados!

FIGURA 3.16 OPERAÇÃO DO `rdt3.0`, O PROTOCOLO BIT ALTERNANTE

3.4.2 Protocolos de transferência confiável de dados com paralelismo

O protocolo `rdt3.0` é correto em termos funcionais, mas é pouco provável que alguém fique contente com o desempenho dele, ainda mais nas redes de alta velocidade de hoje. No coração do problema do desempenho do `rdt3.0` está o fato de ele ser um protocolo do tipo pare e espere.

Para avaliar o impacto sobre o desempenho causado pelo comportamento "pare e espere", considere um caso ideal de dois hospedeiros, um localizado na Costa Oeste dos Estados Unidos e outro na Costa Leste, como mostra a Figura 3.17. O atraso de propagação de ida e volta à velocidade da luz, RTT, entre esses dois sistemas finais é de cerca de 30 ms. Suponha que eles estejam conectados por um canal com capacidade de transmissão, *R*, de 1 Gbit/s (10^9 bits por segundo). Para um tamanho de pacote, *L*, de mil bytes (8 mil bits), incluindo o campo de cabeçalho e também o de dados, o tempo necessário para realmente transmitir o pacote para o enlace de 1 Gbit/s é:

$$t_{trans} = \frac{L}{R} = \frac{8.000 \text{ bits/pacote}}{10^9 \text{ bits/s}} = 8 \mu s$$

A Figura 3.18(a) mostra que, com nosso protocolo pare e espere, se o remetente começar a enviar o pacote em $t = 0$, então em $t = L/R = 8$ µs, o último bit entrará no canal do lado remetente. O pacote então faz sua jornada de 15 ms atravessando o país, com o último bit do pacote emergindo no destinatário em $t = \text{RTT}/2 + L/R = 15{,}008$ ms. Supondo, para simplificar, que pacotes ACK sejam extremamente pequenos (para podermos ignorar seu tempo de transmissão) e que o destinatário pode enviar um ACK logo que receber o último bit de um pacote de dados, o ACK emergirá de volta no remetente em $t = \text{RTT} + L/R = 30{,}008$ ms. Nesse ponto, o remetente agora poderá transmitir a próxima mensagem. Assim, em 30,008 ms, o remetente esteve enviando por apenas 0,008 ms. Se definirmos a ***utilização*** do remetente (ou do canal) como a fração de tempo em que o remetente está realmente ocupado enviando bits para dentro do canal, a análise da Figura 3.18(a) mostra que o protocolo pare e espere tem uma utilização do remetente U_{remet} bastante desanimadora, de:

$$U_{remet} = \frac{L/R}{RTT + L/R} = \frac{0{,}008}{30{,}008} = 0{,}00027$$

Portanto, o remetente ficou ocupado apenas 2,7 centésimos de 1% do tempo! Visto de outra maneira, ele só foi capaz de enviar 1.000 bytes em 30,008 ms, uma vazão efetiva de apenas 267 kbits/s — mesmo estando disponível um enlace de 1 gigabit por segundo! Imagine o infeliz administrador de redes que acabou de pagar uma fortuna para ter capacidade de enlace da ordem de gigabits, mas consegue uma vazão de apenas 267 Kb por segundo! Este é um exemplo gráfico de como protocolos de rede podem limitar as capacidades oferecidas pelo hardware subjacente de rede. Além disso, desprezamos também os tempos de processamento de protocolo das camadas inferiores no remetente e no destinatário, bem como os atrasos de processamento e de fila que ocorre-

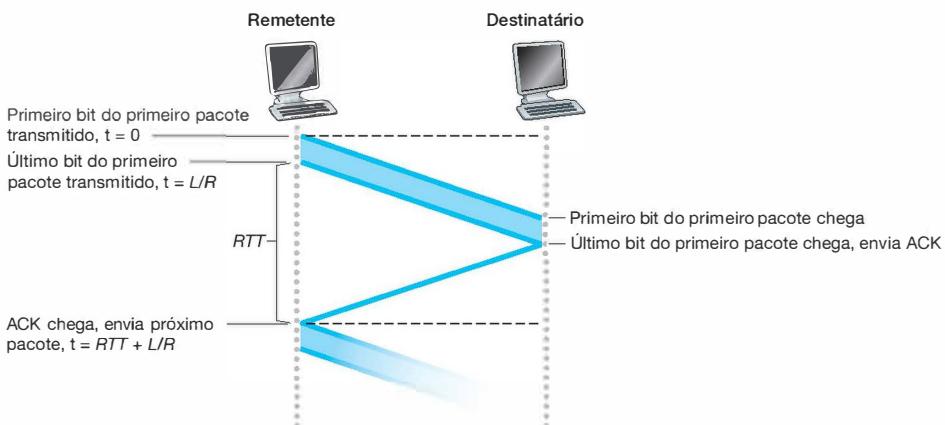
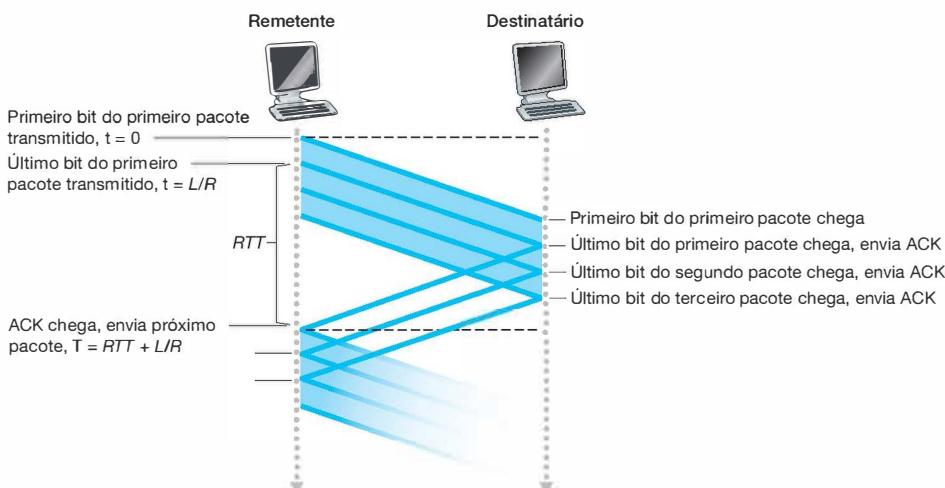



FIGURA 3.17 PROTOCOLO PARE E ESPERE *VERSUS* PROTOCOLO COM PARALELISMO

a. Um protocolo pare e espere em operação

b. Um protocolo com paralelismo em operação

FIGURA 3.18 ENVIO COM PARE E ESPERE E COM PARALELISMO

a. Operação pare e espere

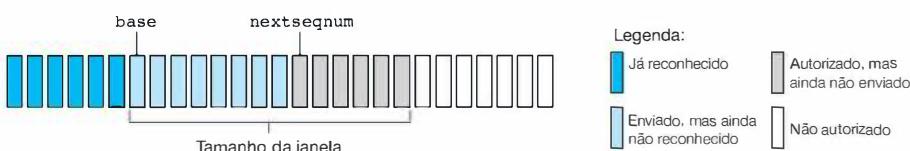
b. Operação com paralelismo

riam em quaisquer roteadores intermediários existentes entre o remetente e o destinatário. Incluir esses efeitos serviria apenas para aumentar ainda mais o atraso e piorar ainda mais o fraco desempenho.

A solução para esse problema de desempenho em especial é simples: em vez de operar em modo pare e espere, o remetente é autorizado a enviar vários pacotes sem esperar por reconhecimentos, como mostra a Figura 3.17(b). A Figura 3.18(b) mostra que, se um remetente for autorizado a transmitir três pacotes antes de ter de esperar por reconhecimentos, sua utilização será triplicada. Uma vez que os muitos pacotes em trânsito entre remetente e destinatário podem ser visualizados como se estivessem enchendo uma tubulação, essa técnica é conhecida, em inglês, como ***pipelining**** (tubulação). O paralelismo gera as seguintes consequências para protocolos de transferência confiável de dados:

* Porém, como essa expressão é difícil de traduzir para o português, preferimos usar "paralelismo", embora a transmissão de dados seja de fato sequencial. (N. do T.)

- A faixa de números de sequência tem de ser ampliada, pois cada pacote em trânsito (sem contar as retransmissões) precisa ter um número de sequência exclusivo e pode haver vários pacotes não reconhecidos em trânsito.
- Os lados remetente e destinatário dos protocolos podem ter de reservar buffers para mais de um pacote. No mínimo, o remetente terá de providenciar buffers para pacotes que foram transmitidos, mas que ainda não foram reconhecidos. O buffer de pacotes bem recebidos pode também ser necessário no destinatário, como discutiremos a seguir.
- A faixa de números de sequência necessária e as necessidades de buffer dependerão da maneira como um protocolo de transferência de dados responde a pacotes perdidos, corrompidos e demasiadamente atrasados. Duas abordagens básicas em relação à recuperação de erros com paralelismo podem ser identificadas: **Go-Back-N** e **repetição seletiva**.

3.4.3 Go-Back-N (GBN)

Em um **protocolo *Go-Back-N* (GBN)**, o remetente é autorizado a transmitir múltiplos pacotes (se disponíveis) sem esperar por um reconhecimento, mas fica limitado a ter não mais do que algum número máximo permitido, **N**, de pacotes não reconhecidos na "tubulação". Nesta seção, descreveremos o protocolo GBN com detalhes. Mas antes de continuar a leitura, convidamos você para se divertir com o applet GBN (incrível!) no site de apoio do livro.

A Figura 3.19 mostra a visão que o remetente tem da faixa de números de sequência em um protocolo GBN. Se definirmos base como o número de sequência do mais antigo pacote não reconhecido e `nextseqnum` como o menor número de sequência não utilizado (isto é, o número de sequência do próximo pacote a ser enviado), então quatro intervalos na faixa de números de sequência poderão ser identificados. Os números de sequência no intervalo `[0,base-1]` correspondem aos pacotes que já foram transmitidos e reconhecidos. O intervalo `[base,nextseqnum-1]` corresponde aos pacotes enviados, mas ainda não foram reconhecidos. Os números de sequência no intervalo `[nextseqnum,base+N-1]` podem ser usados para pacotes que podem ser enviados imediatamente, caso cheguem dados vindos da camada superior. Por fim, números de sequência maiores ou iguais a `base+N` não podem ser usados até que um pacote não reconhecido que esteja pendente seja reconhecido (especificamente, o pacote cujo número de sequência é base).

Como sugere a Figura 3.19, a faixa de números de sequência permitidos para pacotes transmitidos, porém ainda não reconhecidos pode ser vista como uma janela de tamanho *N* sobre a faixa de números de sequência. À medida que o protocolo opera, a janela se desloca para a frente sobre o espaço de números de sequência. Por essa razão, *N* é muitas vezes denominado **tamanho de janela** e o protocolo GBN em si, **protocolo de janela deslizante (*sliding-window protocol*)**. É possível que você esteja pensando que razão teríamos, primeiro, para limitar o número de pacotes pendentes não reconhecidos a um valor *N*. Por que não permitir um número ilimitado deles? Veremos na Seção 3.5 que o controle de fluxo é uma das razões para impor um limite ao remetente. Examinaremos outra razão para isso na Seção 3.7, quando estudarmos o controle de congestionamento do TCP.

Na prática, o número de sequência de um pacote é carregado em um campo de comprimento fixo no cabeçalho do pacote. Se *k* for o número de bits no campo de número de sequência do pacote, a faixa de números de sequência será então $[0,2^k - 1]$. Com uma faixa finita de números de sequência, toda a aritmética que envolver números de sequência deverá ser feita usando aritmética de módulo 2^k. (Em outras palavras, o espaço do número

FIGURA 3.19 VISÃO DO REMETENTE PARA OS NÚMEROS DE SEQUÊNCIA NO PROTOCOLO GO-BACK-N

de sequência pode ser imaginado como um anel de tamanho 2^k, em que o número de sequência $2^k - 1$ é seguido de imediato pelo número de sequência 0.) Lembre-se de que `rdt3.0` tem um número de sequência de 1 bit e uma faixa de números de sequência de [0,1]. Vários problemas ao final deste capítulo tratam das consequências de uma faixa finita de números de sequência. Veremos na Seção 3.5 que o TCP tem um campo de número de sequência de 32 bits, em que os números de sequência do TCP contam bytes na cadeia de bytes, em vez de pacotes.

As figuras 3.20 e 3.21 descrevem uma FSM estendida dos lados remetente e destinatário de um protocolo GBN baseado em ACK, mas sem NAK. Referimo-nos a essa descrição de FSM como *FSM estendida* porque adicionamos variáveis (semelhantes às variáveis de linguagem de programação) para base e nextseqnum; também adicionamos operações sobre essas variáveis e ações condicionais que as envolvem. Note que a especificação da FSM estendida agora está começando a parecer um pouco com uma especificação de linguagem de programação. Bochman [1984] fornece um excelente levantamento sobre extensões adicionais às técnicas FSM, bem como sobre outras técnicas para especificação de protocolos baseadas em linguagens.

FIGURA 3.20 DESCRIÇÃO DA FSM ESTENDIDA DO REMETENTE GBN

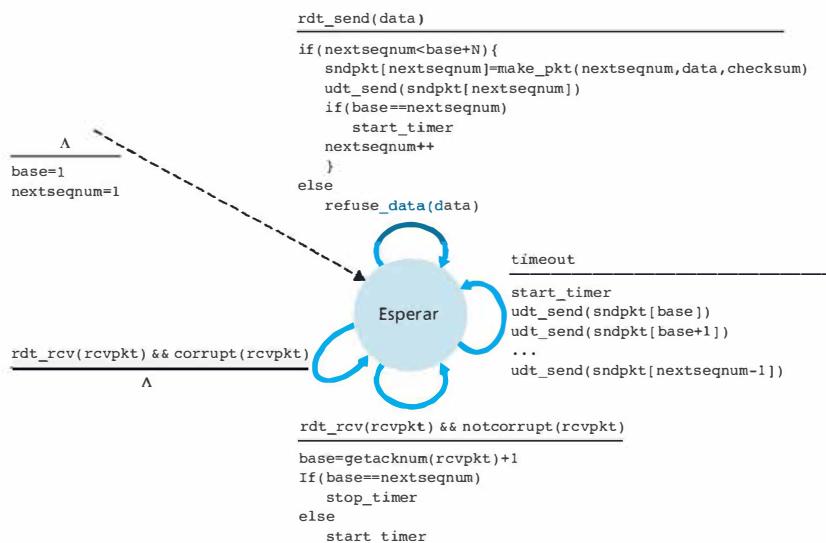
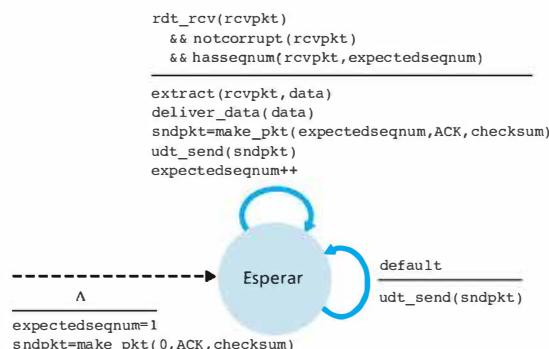
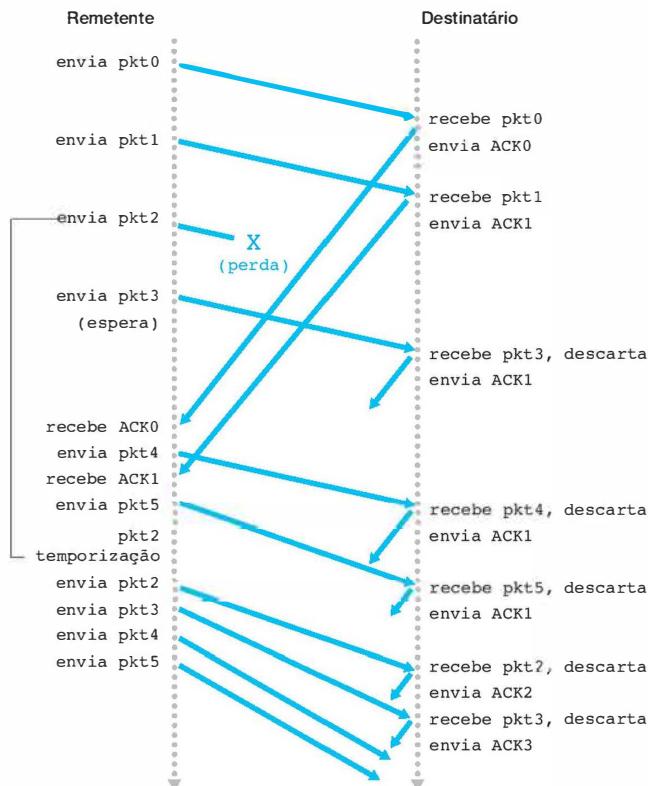



FIGURA 3.21 DESCRIÇÃO DA FSM ESTENDIDA DO DESTINATÁRIO GBN

O remetente GBN deve responder a três tipos de eventos:

- *Chamada vinda de cima.* Quando `rdt_send()` é chamado de cima, o remetente primeiro verifica se a janela está cheia, isto é, se há *N* pacotes pendentes não reconhecidos. Se a janela não estiver cheia, um pacote é criado e enviado e as variáveis são adequadamente atualizadas. Se estiver cheia, o remetente apenas devolve os dados à camada superior — uma indicação implícita de que a janela está cheia. Presume-se que a camada superior então teria de tentar outra vez mais tarde. Em uma execução real, o remetente muito provavelmente teria colocado esses dados em um buffer (mas não os teria enviado imediatamente) ou teria um mecanismo de sincronização (por exemplo, um semáforo ou uma *flag*) que permitiria que a camada superior chamasse `rdt_send()` apenas quando as janelas não estivessem cheias.
- *Recebimento de um ACK.* Em nosso protocolo GBN, um reconhecimento de pacote com número de sequência *n* seria tomado como um **reconhecimento cumulativo**, indicando que todos os pacotes com número de sequência até e inclusive *n* tinham sido corretamente recebidos no destinatário. Voltaremos a esse assunto em breve, quando examinarmos o lado destinatário do GBN.
- *Um evento de esgotamento de temporização (timeout).* O nome "Go-Back-N" deriva do comportamento do remetente em relação a pacotes perdidos ou demasiadamente atrasados. Como no protocolo pare e espere, um temporizador é usado para recuperar a perda de dados ou reconhecer pacotes. Se ocorrer o esgotamento da temporização, o remetente reenvia *todos* os pacotes que tinham sido previamente enviados, mas que ainda não tinham sido reconhecidos. Nosso remetente da Figura 3.20 usa apenas um único temporizador, que pode ser imaginado como um temporizador para o mais antigo pacote já transmitido porém ainda não reconhecido. Se for recebido um ACK e ainda houver pacotes adicionais transmitidos mas ainda não reconhecidos, o temporizador será reiniciado. Se não houver nenhum pacote pendente não reconhecido, o temporizador será desligado.

As ações do destinatário no GBN também são simples. Se um pacote com número de sequência *n* for recebido corretamente e estiver na ordem (isto é, os últimos dados entregues à camada superior vierem de um pacote com número de sequência $n - 1$), o destinatário enviará um ACK para o pacote *n* e entregará a parte de dados do pacote à camada superior. Em todos os outros casos, o destinatário descarta o pacote e reenvia um ACK para o mais recente que foi recebido na ordem correta. Dado que são entregues à camada superior um por vez, se o pacote *k* tiver sido recebido e entregue, então todos os pacotes com número de sequência menores do que *k* também terão sido entregues. Assim, o uso de reconhecimentos cumulativos é uma escolha natural para o GBN.

Em nosso protocolo GBN, o destinatário descarta os pacotes que chegam fora de ordem. Embora pareça bobagem e perda de tempo descartar um pacote corretamente recebido (mas fora de ordem), existem justificativas para isso. Lembre-se de que o destinatário deve entregar dados na ordem certa à camada superior. Suponha agora que o pacote *n* esteja sendo esperado, mas quem chega é o pacote $n + 1$. Como os dados devem ser entregues na ordem certa, o destinatário *poderia* conservar o pacote $n + 1$ no buffer (salvá-lo) e entregá-lo à camada superior mais tarde, após ter recebido o pacote *n*. Contudo, se o pacote *n* for perdido, *n* e $n + 1$ serão ambos por fim retransmitidos como resultado da regra de retransmissão do GBN no remetente. Assim, o destinatário pode apenas descartar o pacote $n + 1$. A vantagem dessa abordagem é a simplicidade da manipulação de buffers no destinatário — ele não precisa colocar no buffer nenhum pacote que esteja fora de ordem. Desse modo, enquanto o remetente deve manter os limites superior e inferior de sua janela e a posição de nextseqnum dentro dela, a única informação que o destinatário precisa manter é o número de sequência do próximo pacote esperado conforme a ordem. Esse valor é retido na variável expectedseqnum mostrada na FSM destinatária da Figura 3.21. Claro, a desvantagem de jogar fora um pacote recebido corretamente é que a retransmissão subsequente desse pacote pode ser perdida ou ficar truncada, caso em que ainda mais retransmissões seriam necessárias.

A Figura 3.22 mostra a operação do protocolo GBN para o caso de um tamanho de janela de quatro pacotes. Por causa da limitação do tamanho dessa janela, o remetente envia os pacotes de 0 a 3, mas, em seguida, tem de esperar que um ou mais desses pacotes sejam reconhecidos antes de prosseguir. E, à medida que cada ACK sucessivo (por exemplo, ACK0 e ACK1) é recebido, a janela se desloca para a frente e o remetente pode transmitir um novo pacote (pkt4 e pkt5, respectivamente). Do lado destinatário, o pacote 2 é perdido. Desse modo, verifica-se que os pacotes 3, 4 e 5 estão fora de ordem e, portanto, são descartados.

FIGURA 3.22 GO-BACK-N EM OPERAÇÃO

Antes de encerrarmos nossa discussão sobre o GBN, devemos ressaltar que uma implementação desse protocolo em uma pilha de protocolo provavelmente seria estruturada de modo semelhante à da FSM estendida da Figura 3.20. A implementação também seria estruturada sob a forma de vários procedimentos que implementam as ações a serem executadas em resposta aos vários eventos que podem ocorrer. Nessa **programação baseada em eventos**, os vários procedimentos são chamados (invocados) por outros procedimentos presentes na pilha de protocolo ou como resultado de uma interrupção. No remetente, seriam: (1) uma chamada pela entidade da camada superior invocando `rdt_send()`, (2) uma interrupção pelo temporizador e (3) uma chamada pela camada inferior invocando `rdt_rcv()` quando chega um pacote. Os exercícios de programação ao final deste capítulo lhe darão a chance de executar de verdade essas rotinas em um ambiente de rede simulado, mas realista.

Salientamos que o protocolo GBN incorpora quase todas as técnicas que encontraremos quando estudarmos, na Seção 3.5, os componentes de transferência confiável de dados do TCP. Essas técnicas incluem a utilização de números de sequência, reconhecimentos cumulativos, somas de verificação e uma operação de esgotamento de temporização/retransmissão.

3.4.4 Repetição seletiva (SR)

O protocolo GBN permite que o remetente potencialmente "encha a rede" com pacotes na Figura 3.17, evitando, assim, os problemas de utilização de canal observados em protocolos do tipo pare e espere. Há, contudo,

casos em que o próprio GBN sofre com problemas de desempenho. Em especial, quando o tamanho da janela e o produto entre o atraso e a largura de banda são grandes, pode haver muitos pacotes pendentes na rede. Assim, um único erro de pacote pode fazer que o GBN retransmita um grande número de pacotes — muitos deles sem necessidade. À medida que aumenta a probabilidade de erros no canal, a rede pode ficar lotada com essas retransmissões desnecessárias. Imagine se, em nosso cenário de conversa, toda vez que uma palavra fosse pronunciada de maneira truncada as outras mil que a circundam (por exemplo, um tamanho de janela de mil palavras) tivessem de ser repetidas. A conversa sofreria atrasos por causa de todas essas palavras reiteradas.

Como o próprio nome sugere, protocolos de repetição seletiva (*selective repeat* — SR) evitam retransmissões desnecessárias porque fazem o remetente retransmitir apenas os pacotes suspeitos de terem sido recebidos com erro (isto é, que foram perdidos ou corrompidos) no destinatário. Essa retransmissão individual, só quando necessária, exige que o destinatário reconheça *individualmente* os pacotes recebidos de modo correto. Uma janela de tamanho *N* será usada novamente para limitar o número de pacotes pendentes não reconhecidos dentro da rede. Contudo, ao contrário do GBN, o remetente já terá recebido ACKs para alguns dos pacotes na janela. A Figura 3.23 mostra a visão que o protocolo de SR remetente tem do espaço do número de sequência; a Figura 3.24 detalha as várias ações executadas pelo protocolo SR remetente.

FIGURA 3.23 VISÕES QUE OS PROTOCOLOS SR REMETENTE E DESTINATÁRIO TÊM DO ESPAÇO DE NÚMERO DE SEQUÊNCIA

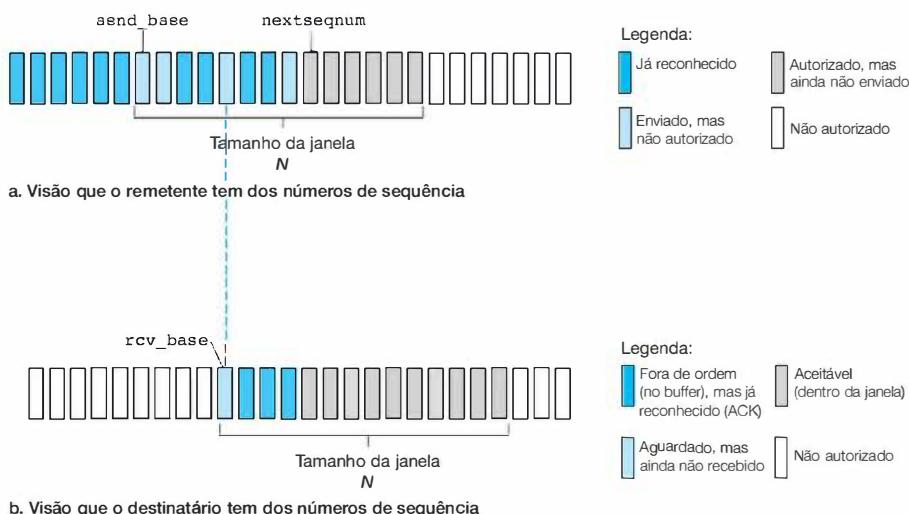
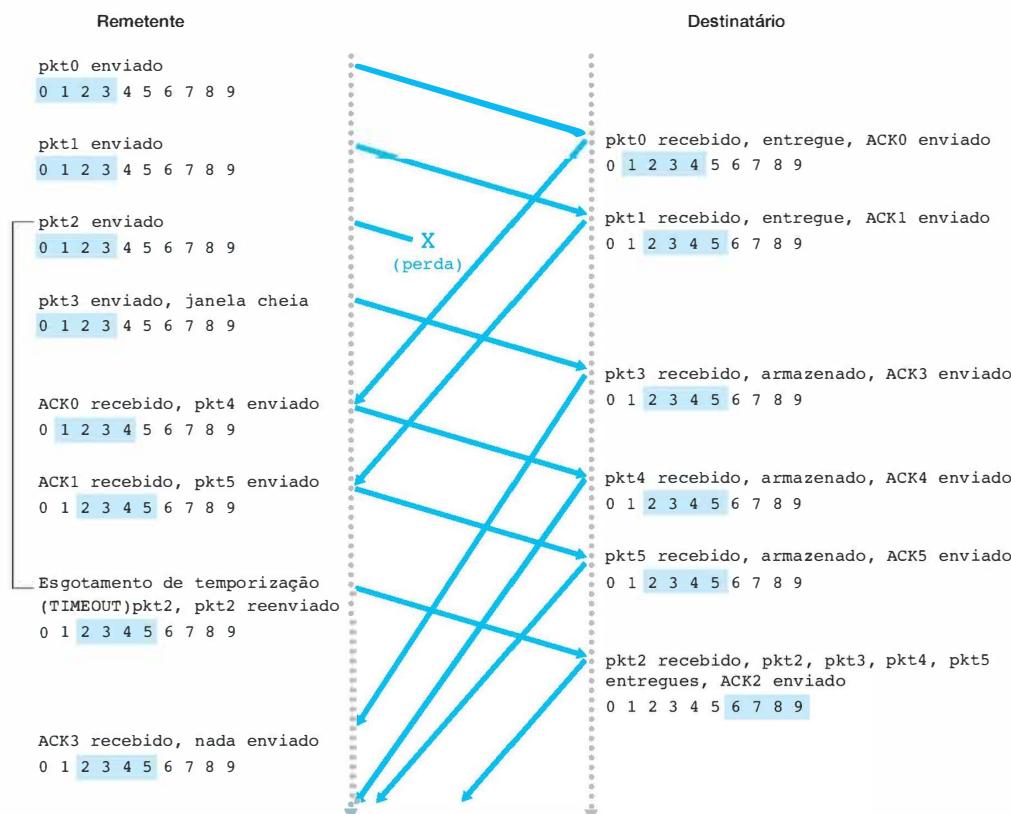



FIGURA 3.24 EVENTOS E AÇÕES DO PROTOCOLO SR REMETENTE

1. *Dados recebidos de cima.* Quando são recebidos dados de cima, o protocolo SR remetente verifica o próximo número de sequência disponível para o pacote. Se o número de sequência está dentro da janela do remetente, os dados são empacotados e enviados; do contrário, eles são armazenados ou devolvidos à camada superior para transmissão posterior, como acontece no GBN.

1. *Esgotamento de temporização.* Novamente são usados temporizadores para proteção contra perda de pacotes. Contudo, cada pacote agora deve ter seu próprio temporizador lógico, já que apenas um pacote será transmitido quando a temporização se esgotar. Um único hardware de temporizador pode ser usado para emular a operação de múltiplos temporizadores lógicos [Varghese, 1997].

2. *ACK recebido.* Se for recebido um ACK, o SR remetente marcará aquele pacote como recebido, contanto que esteja na janela. Se o número de sequência do pacote for igual a `send_base`, a base da janela se deslocará para a frente até o pacote não reconhecido que tiver o menor número de sequência. Se a janela se deslocar e houver pacotes não transmitidos com números de sequência que agora caem dentro da janela, esses pacotes serão transmitidos.

O protocolo SR destinatário reconhecerá um pacote corretamente recebido esteja ele ou não na ordem certa. Pacotes fora de ordem ficam no buffer até que todos os faltantes (isto é, os que têm números de sequência menores) sejam recebidos, quando então um conjunto de pacotes poderá ser entregue à camada superior na ordem correta. A Figura 3.25 apresenta as várias ações realizadas pelo protocolo SR destinatário. A Figura 3.26 mostra um exemplo de operação do protocolo SR quando ocorre perda de pacotes. Note que, nessa figura, o destinatário de início armazena os pacotes 3, 4 e 5 e os entrega junto com o pacote 2 à camada superior, quando o pacote 2 é enfim recebido.

FIGURA 3.25 EVENTOS E AÇÕES DO PROTOCOLO SR DESTINATÁRIO

1. *Pacote com número de sequência no intervalo* [rcv_base, rcv_base+N–1] *foi corretamente recebido.* Nesse caso, o pacote recebido cai dentro da janela do destinatário e um pacote ACK seletivo é devolvido ao remetente. Se o pacote não tiver sido recebido anteriormente, irá para o buffer. Se esse pacote tiver um número de sequência igual à base da janela destinatária (rcv_base na Figura 3.22), então ele e quaisquer outros pacotes armazenados no buffer e numerados consecutivamente (começando com rcv_base) serão entregues à camada superior. A janela destinatária é então deslocada para a frente de acordo com o número de pacotes entregues à camada superior. Como exemplo, considere a Figura 3.26. Quando um pacote com número de sequência rcv_base=2 é recebido, ele e os pacotes 3, 4 e 5 podem ser entregues à camada superior.

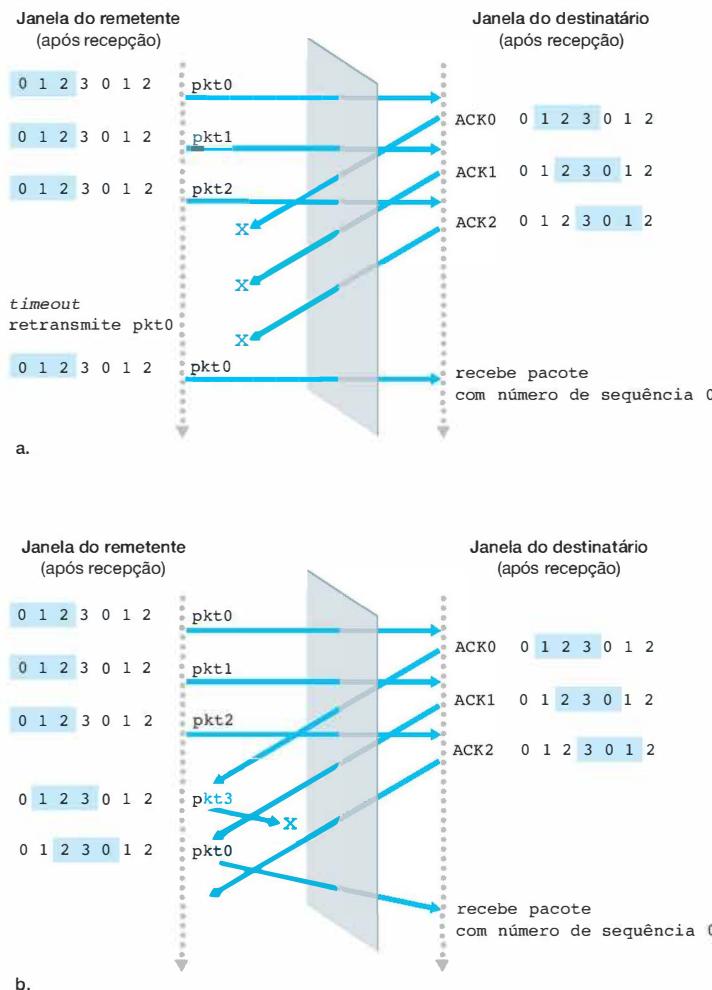
1. *Pacote com número de sequência no intervalo* [rcv_base-N, rcv_base-1] *foi corretamente recebido.* Nesse caso, um ACK deve ser gerado mesmo que esse pacote já tenha sido reconhecido pelo destinatário.

2. *Qualquer outro.* Ignore o pacote.

FIGURA 3.26 OPERAÇÃO SR

É importante notar que na etapa 2 da Figura 3.25 o destinatário reconhece novamente (em vez de ignorar) pacotes já recebidos com certos números de sequência que estão *abaixo* da atual base da janela. É bom que você se convença de que esse reconhecimento duplo é de fato necessário. Dados os espaços dos números de sequência do remetente e do destinatário na Figura 3.23, por exemplo, se não houver ACK para pacote com número `send_base` propagando-se do destinatário ao remetente, este acabará retransmitindo o pacote `send_base`, embora esteja claro (para nós, e não para o remetente!) que o destinatário já o recebeu. Caso o destinatário não o reconhecesse, a janela do remetente jamais se deslocaria para a frente! Esse exemplo ilustra um importante aspecto dos protocolos SR (e também de muitos outros). O remetente e o destinatário nem sempre têm uma visão idêntica do que foi recebido corretamente e do que não foi. Para protocolos SR, isso significa que as janelas do remetente e do destinatário nem sempre coincidirão.

A falta de sincronização entre as janelas do remetente e do destinatário tem importantes consequências quando nos defrontamos com a realidade de uma faixa finita de números de sequência. Considere o que poderia acontecer, por exemplo, com uma faixa finita de quatro números de sequência de pacotes (0, 1, 2, 3) e um tamanho de janela de três. Suponha que os pacotes de 0 a 2 sejam transmitidos, recebidos e reconhecidos corretamente no destinatário. Nesse ponto, a janela do destinatário está sobre o quarto, o quinto e o sexto pacotes, que têm os números de sequência 3, 0 e 1. Agora, considere dois cenários. No primeiro, mostrado na Figura 3.27(a), os ACKs para os três primeiros pacotes foram perdidos e o remetente os retransmite. Assim, o que o destinatário recebe em seguida é um pacote com o número de sequência 0 — uma cópia do primeiro pacote enviado.

No segundo cenário, mostrado na Figura 3.27(b), os ACKs para os três primeiros pacotes foram entregues de modo correto. Assim, o remetente desloca sua janela para a frente e envia o quarto, o quinto e o sexto pacotes com os números de sequência 3, 0 e 1, respectivamente. O pacote com o número de sequência 3 é perdido, mas o pacote com o número de sequência 0 chega — um pacote que contém dados *novos*.

Agora, na Figura 3.27, considere o ponto de vista do destinatário, que tem uma cortina imaginária entre o remetente e ele, já que o destinatário não pode "ver" as ações executadas pelo remetente. Tudo o que o destinatário observa é a sequência de mensagens que ele recebe do canal e envia para o canal. No que lhe concerne, os dois cenários da Figura 3.27 são *idênticos*. Não há um modo de distinguir a retransmissão do primeiro pacote da transmissão original do quinto pacote. Fica claro que um tamanho de janela que seja igual ao tamanho do espaço de numeração sequencial menos 1 não vai funcionar. Mas qual deve ser o tamanho da janela? Um problema ao final deste capítulo pede que você demonstre que o tamanho pode ser menor ou igual à metade do tamanho do espaço de numeração sequencial para os protocolos SR.

No site de apoio do livro, você encontrará um applet que anima a operação do protocolo SR. Tente realizar os mesmos experimentos feitos com o applet GBN. Os resultados combinam com o que você espera?

Isso encerra nossa discussão sobre protocolos de transferência confiável de dados. Percorremos um *longo* caminho e apresentamos numerosos mecanismos que, juntos, proveem transferência confiável de dados. A Tabela 3.1 resume esses mecanismos. Agora que já vimos todos eles em operação e podemos enxergar "o quadro geral", aconselhamos que você leia novamente esta seção para perceber como esses mecanismos foram adicionados pouco a pouco, de modo a abordar modelos (realistas) de complexidade crescente do canal que conecta o remetente ao destinatário ou para melhorar o desempenho dos protocolos.

Encerraremos nossa explanação considerando uma premissa remanescente em nosso modelo de canal subjacente. Lembre-se de que admitimos que pacotes não podem ser reordenados dentro do canal entre o remetente e o destinatário. Esta é uma premissa em geral razoável quando o remetente e o destinatário estão conectados por um único fio físico. Contudo, quando o "canal" que conecta os dois é uma rede, pode ocorrer reordenação de pacotes. Uma manifestação da reordenação de pacotes é que podem aparecer cópias antigas de um pacote com número de sequência ou de reconhecimento x, mesmo que nem a janela do remetente nem a do destinatário contenham x. Com a reordenação de pacotes, podemos considerar que o canal usa armazenamento de pacotes e emite-os espontaneamente em algum momento qualquer do futuro. Como números de sequência podem ser reutilizados, devemos tomar algum cuidado para nos prevenir contra esses pacotes duplicados. A abordagem adotada na prática é garantir que um número de sequência não seja reutilizado até que o remetente esteja "certo" de que nenhum pacote enviado antes com número de sequência x está na rede. Isso é feito admitindo que um pacote não pode "viver" na

FIGURA 3.27 DILEMA DO REMETENTE SR COM JANELAS MUITO GRANDES: UM NOVO PACOTE OU UMA RETRANSMISSÃO?

rede mais do que um tempo máximo fixado. As extensões do TCP para redes de alta velocidade [RFC 1323] usam um tempo de vida máximo de pacote de cerca de três minutos. Sunshine [1978] descreve um método para usar números de sequência tais que os problemas de reordenação podem ser completamente evitados.

3.5 TRANSPORTE ORIENTADO PARA CONEXÃO: TCP

Agora que já vimos os princípios subjacentes à transferência confiável de dados, vamos voltar para o TCP — o protocolo de transporte confiável da camada de transporte, orientado para conexão, da Internet. Nesta seção, veremos que, para poder fornecer transferência confiável de dados, o TCP conta com muitos dos princípios subjacentes

TABELA 3.1 RESUMO DE MECANISMOS DE TRANSFERÊNCIA CONFIÁVEL DE DADOS E SUA UTILIZAÇÃO

Mecanismo	Uso, comentários
Soma de verificação	Usada para detectar erros de bits em um pacote transmitido.
Temporizador	Usado para controlar a temporização/retransmissão de um pacote, possivelmente porque o pacote (ou seu ACK) foi perdido dentro do canal. Como pode ocorrer esgotamento de temporização quando um pacote está atrasado, mas não perdido (esgotamento de temporização prematuro), ou quando um pacote foi recebido pelo destinatário mas o ACK remetente-destinatário foi perdido, um destinatário pode receber cópias duplicadas de um pacote.
Número de sequência	Usado para numeração sequencial de pacotes de dados que transitam do remetente ao destinatário. Lacunas nos números de sequência de pacotes recebidos permitem que o destinatário detecte um pacote perdido. Pacotes com números de sequência duplicados permitem que o destinatário detecte cópias duplicadas de um pacote.
Reconhecimento	Usado pelo destinatário para avisar o remetente que um pacote ou conjunto de pacotes foi recebido corretamente. Reconhecimentos normalmente portam o número de sequência do pacote, ou pacotes, que estão sendo reconhecidos. Reconhecimentos podem ser individuais ou cumulativos, dependendo do protocolo.
Reconhecimento negativo	Usado pelo destinatário para avisar o remetente que um pacote não foi recebido corretamente. Reconhecimentos negativos normalmente portam o número de sequência do pacote que não foi recebido corretamente.
Janela, paralelismo	O remetente pode ficar restrito a enviar somente pacotes com números de sequência que caiam dentro de uma determinada faixa. Permitindo que vários pacotes sejam transmitidos, ainda que não reconhecidos, a utilização do remetente pode ser aumentada em relação ao modo de operação pare e espere. Em breve veremos que o tamanho da janela pode ser estabelecido com base na capacidade de o destinatário receber e fazer buffer de mensagens ou no nível de congestionamento na rede, ou em ambos.

discutidos na seção anterior, incluindo detecção de erro, retransmissões, reconhecimentos cumulativos, temporizadores e campos de cabeçalho para números de sequência e de reconhecimento. O TCP está definido nos RFCs 793, 1122, 1323, 2018 e 2581.

3.5.1 A conexão TCP

Dizemos que o TCP é **orientado para conexão** porque, antes que um processo de aplicação possa começar a enviar dados a outro, os dois processos precisam primeiro se "apresentar" — isto é, devem enviar alguns segmentos preliminares um ao outro para estabelecer os parâmetros da transferência de dados. Como parte do estabelecimento da conexão TCP, ambos os lados da conexão iniciarão muitas variáveis de estado (muitas das quais serão discutidas nesta seção e na Seção 3.7) associadas com a conexão TCP.

A "conexão" TCP não é um circuito TDM ou FDM fim a fim, como acontece em uma rede de comutação de circuitos. Tampouco é um circuito virtual (veja o Capítulo 1), pois o estado de conexão reside inteiramente nos dois sistemas finais. Como o protocolo TCP roda apenas nesses sistemas, e não nos elementos intermediários da rede (roteadores e comutadores — *switches* — de camada de enlace), os elementos intermediários não mantêm estado de conexão TCP. Na verdade, os roteadores intermediários são de todo alheios às conexões TCP; eles enxergam datagramas, e não conexões.

Uma conexão TCP provê um **serviço *full-duplex***: se houver uma conexão TCP entre o processo A em um hospedeiro e o processo B em outro hospedeiro, os dados da camada de aplicação poderão fluir de A para B ao mesmo tempo em que os dados da camada de aplicação fluem de B para A. A conexão TCP é sempre **ponto a ponto**, isto é, entre um único remetente e um único destinatário. O chamado "*multicast*" (veja a Seção 4.7) — a transferência de dados de um remetente para vários destinatários em uma única operação de envio — não é possível com o TCP. Com o TCP, dois hospedeiros é bom; três é demais!

Vamos agora examinar como uma conexão TCP é estabelecida. Suponha que um processo que roda em um hospedeiro queira iniciar a conexão com outro processo em outro hospedeiro. Lembre-se de que o processo que está iniciando a conexão é denominado *processo cliente*, e o outro é denominado *processo servidor*. O processo de aplicação cliente primeiro informa à camada de transporte no cliente que ele quer estabelecer uma conexão com um processo no servidor. Lembre-se (Seção 2.7.2) de que um programa cliente em Python faz isso emitindo o comando

```
clientSocket.connect((serverName,serverPort))
```

em que `servername` é o nome do servidor e `serverPort` identifica o processo no servidor. O TCP no cliente então passa a estabelecer uma conexão TCP com o TCP no servidor. Discutiremos com algum detalhe o procedimento de estabelecimento de conexão ao final desta seção. Por enquanto, basta saber que o cliente primeiro envia um segmento TCP especial; o servidor responde com um segundo segmento TCP especial e, por fim, o cliente responde novamente com um terceiro segmento especial. Os primeiros dois segmentos não contêm nenhuma "carga útil", isto é, nenhum dado da camada de aplicação; o terceiro pode carregar uma carga útil. Como três segmentos são enviados entre dois hospedeiros, esse procedimento de estabelecimento de conexão é muitas vezes denominado **apresentação de três vias** (*3-way handshake*).

Uma vez estabelecida uma conexão TCP, os dois processos de aplicação podem enviar dados um para o outro. Vamos considerar o envio de dados do processo cliente para o processo servidor. O processo cliente passa uma cadeia de dados pelo *socket* (a porta do processo), como descrito na Seção 2.7. Tão logo passem pelo *socket*, os dados estão nas mãos do TCP que está rodando no cliente. Como mostra a Figura 3.28, o TCP direciona seus dados para o **buffer de envio** da conexão, que é um dos buffers reservados durante a apresentação de três vias inicial. De quando em quando, o TCP arranca pedaços de dados do buffer de envio e passa os dados à camada de rede. O interessante é que a especificação do TCP [RFC 793] é muito vaga ao indicar quando o TCP deve de fato enviar dados que estão nos buffers, determinando apenas que o TCP "deve enviar aqueles dados em segmentos segundo sua própria conveniência". A quantidade máxima de dados que pode ser retirada e colocada em um segmento é limitada pelo **tamanho máximo do segmento** (*maximum segment size* — MSS). O MSS normalmente é estabelecido determinando primeiro o tamanho do maior quadro de camada de enlace que pode ser enviado pelo hospedeiro remetente local (denominado **unidade máxima de transmissão** — *maximum transmission unit* — MTU) e, em seguida, estabelecendo um MSS que garanta que um segmento TCP (quando encapsulado em um datagrama IP) mais o comprimento do cabeçalho TCP/IP (em geral, 40 bytes) caberão em um único quadro de camada de enlace. Os protocolos da camada de enlace Ethernet e PPP possuem um MSS de 1.500 bytes. Também foram propostas técnicas para descobrir a MTU do caminho — o maior quadro de camada de enlace que pode ser enviado por todos os enlaces desde a origem até o destino [RFC 1191] — e definir o MSS com base no valor da MTU do caminho. Note que o

HISTÓRIA

Vinton Cerf, Robert Kahn e TCP/IP

No início da década de 1970, as redes de comutação de pacotes começaram a proliferar. A ARPAnet — precursora da Internet — era apenas mais uma dentre tantas que tinham, cada uma, seu próprio protocolo. Dois pesquisadores, Vinton Cerf e Robert Kahn, reconheceram a importância de interconectar essas redes e inventaram um protocolo inter-redes denominado TCP/IP, que quer dizer Transmission Control Protocol/Internet Protocol (protocolo de controle de transmissão/protocolo da Internet). Embora no começo Cerf e Kahn considerassem o protocolo uma entidade única, mais tarde ele foi dividido em duas partes, TCP e IP, que operavam em separado. Cerf e Kahn publicaram um artigo sobre o TCP/IP em maio de 1974 em *IEEE Transactions on Communication Technology* [Cerf, 1974].

O protocolo TCP/IP, que é o "feijão com arroz" da Internet de hoje, foi elaborado antes dos PCs, estações de trabalho, smartphones e tablets, antes da proliferação da Ethernet, cabo, DSL, Wi-Fi e outras tecnologias de redes locais, antes da Web, redes sociais e recepção de vídeo. Cerf e Kahn perceberam a necessidade de um protocolo de rede que, de um lado, fornecesse amplo suporte para aplicações ainda a serem definidas e que, de outro, permitisse a interoperação de hospedeiros arbitrários e protocolos de camada de enlace.

Em 2004, Cerf e Kahn receberam o prêmio ACM Turing Award, considerado o Prêmio Nobel da Computação pelo "trabalho pioneiro sobre interligação em rede, incluindo o projeto e a implementação dos protocolos de comunicação da Internet, TCP/IP e por inspirarem liderança na área de redes".

MSS é a quantidade máxima de dados de camada de aplicação no segmento, e não o tamanho máximo do segmento TCP incluindo cabeçalhos. (Essa terminologia é confusa, mas temos de conviver com ela, pois já está arraigada.)

O TCP combina cada porção de dados do cliente com um cabeçalho TCP, formando, assim, **segmentos TCP**. Os segmentos são passados para baixo, para a camada de rede, onde são encapsulados separadamente dentro dos datagramas IP da camada de rede. Os datagramas IP são então enviados para dentro da rede. Quando o TCP recebe um segmento na outra extremidade, os dados do segmento são colocados no buffer de recepção da conexão, como ilustra a Figura 3.28. A aplicação lê a cadeia de dados desse buffer. Cada lado da conexão tem seus próprios buffers de envio e seu próprio buffer de recepção. (Você pode ver o applet de controle de fluxo on-line em <http://www.awl.com/kurose-ross>, que oferece uma animação dos buffers de envio e de recepção.)

Entendemos, dessa discussão, que uma conexão TCP consiste em buffers, variáveis e um *socket* de conexão de um processo em um hospedeiro e outro conjunto de buffers, variáveis e um *socket* de conexão de um processo em outro hospedeiro. Como mencionamos, nenhum buffer nem variáveis são alocados à conexão nos elementos da rede (roteadores, comutadores e repetidores) existentes entre os hospedeiros.

3.5.2 Estrutura do segmento TCP

Agora que examinamos de modo breve a conexão TCP, vamos verificar a estrutura do segmento TCP, que consiste em campos de cabeçalho e um campo de dados. O campo de dados contém uma quantidade de dados de aplicação. Como já dissemos, o MSS limita o tamanho máximo do campo de dados de um segmento. Quando o TCP envia um arquivo grande, tal como uma imagem de uma página Web, ele costuma fragmentar o segmento em pedaços de tamanho MSS (exceto o último, que muitas vezes é menor do que o MSS). Aplicações interativas, contudo, muitas vezes transmitem quantidades de dados menores do que o MSS. Por exemplo, com aplicações de login remoto como Telnet, o campo de dados do segmento TCP é, muitas vezes, de apenas 1 byte. Como o cabeçalho TCP tem tipicamente 20 bytes (12 bytes mais do que o cabeçalho UDP), o comprimento dos segmentos enviados por Telnet pode ser de apenas 21 bytes.

A Figura 3.29 mostra a estrutura do segmento TCP. Como acontece com o UDP, o cabeçalho inclui **números de porta de origem e de destino**, que são usados para multiplexação e demultiplexação de dados de/para aplicações de camadas superiores e, assim como no UDP, inclui um **campo de soma de verificação**. Um cabeçalho de segmento TCP também contém os seguintes campos:

- O **campo de número de sequência** de 32 bits e o **campo de número de reconhecimento** de 32 bits são usados pelos TCPs remetente e destinatário na execução de um serviço confiável de transferência de dados, como discutido a seguir.
- O campo de **janela de recepção** de 16 bits é usado para controle de fluxo. Veremos em breve que esse campo é usado para indicar o número de bytes que um destinatário está disposto a aceitar.

FIGURA 3.28 BUFFERS TCP DE ENVIO E DE RECEPÇÃO

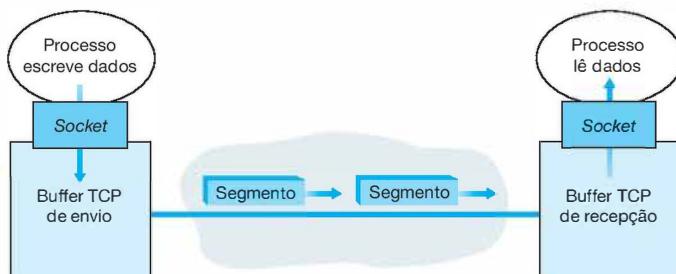
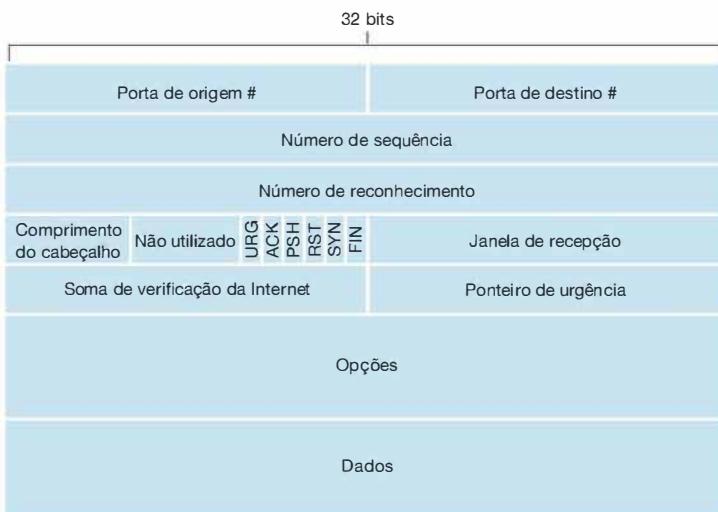



FIGURA 3.29 ESTRUTURA DO SEGMENTO TCP

32 bits

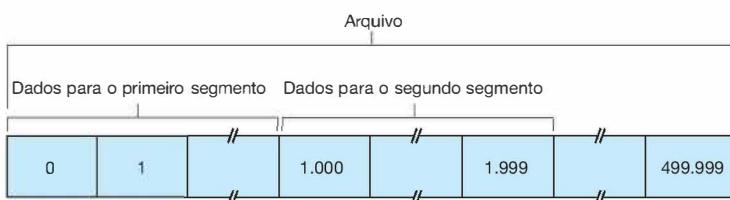
Porta de origem #
Porta de destino #
Número de sequência
Número de reconhecimento
Comprimento do cabeçalho
Não utilizado
URG
ACK
PSH
RST
SYN
FIN
Janela de recepção
Soma de verificação da Internet
Ponteiro de urgência
Opções
Dados

- O **campo de comprimento de cabeçalho** de 4 bits especifica o comprimento do cabeçalho TCP em palavras de 32 bits. O cabeçalho TCP pode ter comprimento variável por causa do campo de opções TCP. (O campo de opções TCP em geral está vazio, de modo que o comprimento do cabeçalho TCP típico é 20 bytes.)
- O **campo de opções**, opcional e de comprimento variável, é usado quando um remetente e um destinatário negociam o MSS, ou como um fator de aumento de escala da janela para utilização em redes de alta velocidade. Uma opção de marca de tempo é também definida. Consulte o RFC 854 e o RFC 1323 para detalhes adicionais.
- O **campo de flag** contém 6 bits. O **bit ACK** é usado para indicar se o valor carregado no campo de reconhecimento é válido, isto é, se o segmento contém um reconhecimento para um segmento que foi recebido com sucesso. Os bits **RST**, **SYN** e **FIN** são usados para estabelecer e encerrar a conexão, como discutiremos ao final desta seção. Marcar o bit **PSH** indica que o destinatário deve passar os dados para a camada superior imediatamente. Por fim, o bit **URG** é usado para mostrar que há dados nesse segmento que a entidade da camada superior do lado remetente marcou como "urgentes". A localização do último byte desses dados urgentes é indicada pelo **campo de ponteiro de urgência** de 16 bits. O TCP deve informar à entidade da camada superior do lado destinatário quando existem dados urgentes e passar a ela um ponteiro para o final desses dados. (Na prática, o PSH, o URG e o ponteiro de dados urgentes não são usados. Contudo, mencionamos esses campos para descrever todos.)

Números de sequência e números de reconhecimento

Dois dos mais importantes campos do cabeçalho do segmento TCP são o de número de sequência e o de número de reconhecimento. Esses campos são parte fundamental do serviço de transferência confiável de dados do TCP. Mas, antes de discutirmos como são utilizados, vamos explicar exatamente o que o TCP coloca nesses campos.

O TCP vê os dados como uma cadeia de bytes não estruturada, mas ordenada. O uso que o TCP faz dos números de sequência reflete essa visão, pois esses números são aplicados sobre a cadeia de bytes transmitidos, e *não* sobre a série de segmentos transmitidos. O **número de sequência para um segmento** é o número do primeiro

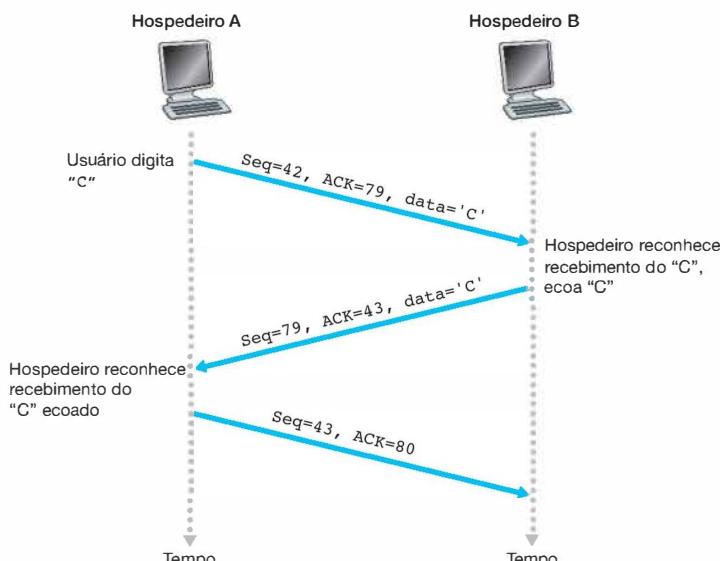
byte do segmento. Vamos ver um exemplo. Suponha que um processo no hospedeiro A queira enviar uma cadeia de dados para um processo no hospedeiro B por uma conexão TCP. O TCP do hospedeiro A vai implicitamente numerar cada byte da cadeia de dados. Suponha que a cadeia de dados consista em um arquivo composto de 500 mil bytes, que o MSS seja de 1.000 bytes e que seja atribuído o número 0 ao primeiro byte da cadeia de dados. Como mostra a Figura 3.30, o TCP constrói 500 segmentos a partir da cadeia de dados. O primeiro recebe o número de sequência 0; o segundo, o número de sequência 1.000; o terceiro, o número de sequência 2.000, e assim por diante. Cada número de sequência é inserido no campo de número de sequência no cabeçalho do segmento TCP apropriado.

Vamos agora considerar os números de reconhecimento. Eles são um pouco mais complicados do que os números de sequência. Lembre-se de que o TCP é *full-duplex*, portanto o hospedeiro A pode estar recebendo dados do hospedeiro B enquanto envia dados ao hospedeiro B (como parte da mesma conexão TCP). Cada segmento que chega do hospedeiro B tem um número de sequência para os dados que estão fluindo de B para A. *O número de reconhecimento que o hospedeiro A atribui a seu segmento é o número de sequência do próximo byte que ele estiver aguardando do hospedeiro B*. É bom examinarmos alguns exemplos para entendermos o que está acontecendo aqui. Suponha que o hospedeiro A tenha recebido do hospedeiro B todos os bytes numerados de 0 a 535 e também que esteja prestes a enviar um segmento ao hospedeiro B. O hospedeiro A está esperando pelo byte 536 e por todos os bytes subsequentes da cadeia de dados do hospedeiro B. Assim, ele coloca o número 536 no campo de número de reconhecimento do segmento que envia para o hospedeiro B.

Como outro exemplo, suponha que o hospedeiro A tenha recebido um segmento do hospedeiro B contendo os bytes de 0 a 535 e outro segmento contendo os bytes de 900 a 1.000. Por alguma razão, o hospedeiro A ainda não recebeu os bytes de 536 a 899. Nesse exemplo, ele ainda está esperando pelo byte 536 (e os superiores) para poder recriar a cadeia de dados de B. Assim, o segmento seguinte que A envia a B conterá 536 no campo de número de reconhecimento. Como o TCP somente reconhece bytes até o primeiro byte que estiver faltando na cadeia, dizemos que o TCP provê **reconhecimentos cumulativos**.

Este último exemplo também revela uma questão importante, mas sutil. O hospedeiro A recebeu o terceiro segmento (bytes de 900 a 1.000) antes do segundo (bytes de 536 a 899). Portanto, o terceiro segmento chegou fora de ordem. E o que um hospedeiro faz quando recebe segmentos fora de ordem em uma conexão TCP? Eis a questão. O interessante é que os RFCs do TCP não impõem nenhuma regra para isso e deixam a decisão para quem estiver programando a execução TCP. Há basicamente duas opções: (1) o destinatário descarta imediatamente os segmentos fora de ordem (o que, como discutimos antes, pode simplificar o projeto do destinatário) ou (2) o destinatário conserva os bytes fora de ordem e espera pelos bytes faltantes para preencher as lacunas. Claro que a segunda alternativa é mais eficiente em termos de largura de banda de rede e é a abordagem adotada na prática.

Na Figura 3.30, admitimos que o número de sequência inicial era 0. Na verdade, ambos os lados de uma conexão TCP escolhem ao acaso um número de sequência inicial. Isso é feito para minimizar a possibilidade de um segmento de uma conexão já encerrada entre dois hospedeiros e ainda presente na rede ser tomado por um segmento válido em uma conexão posterior entre esses dois mesmos hospedeiros (que também podem estar usando os mesmos números de porta da conexão antiga) [Sunshine, 1978].

FIGURA 3.30 DIVIDINDO OS DADOS DO ARQUIVO EM SEGMENTOS TCP

Telnet: um estudo de caso para números de sequência e números de reconhecimento

O Telnet, definido no RFC 854, é um protocolo popular de camada de aplicação utilizado para fazer login remoto. Ele roda sobre TCP e é projetado para trabalhar entre qualquer par de hospedeiros. Diferentemente das aplicações de transferência de dados em grandes blocos, que foram discutidas no Capítulo 2, o Telnet é uma aplicação interativa. Discutiremos, agora, um exemplo de Telnet, pois ilustra muito bem números de sequência e de reconhecimento do TCP. Observamos que muitos usuários agora preferem usar o protocolo SSH, visto que dados enviados por uma conexão Telnet (incluindo senhas!) não são criptografados, o que torna essa aplicação vulnerável a ataques de bisbilhoteiros (como discutiremos na Seção 8.7).

Suponha que o hospedeiro A inicie uma sessão Telnet com o hospedeiro B. Como o hospedeiro A inicia a sessão, ele é rotulado de cliente, enquanto B é rotulado de servidor. Cada caractere digitado pelo usuário (no cliente) será enviado ao hospedeiro remoto; este devolverá uma cópia ("eco") de cada caractere, que será apresentada na tela Telnet do usuário. Esse eco é usado para garantir que os caracteres vistos pelo usuário do Telnet já foram recebidos e processados no local remoto. Assim, cada caractere atravessa a rede duas vezes entre o momento em que o usuário aperta o teclado e o momento em que o caractere é apresentado em seu monitor.

Suponha agora que o usuário digite a letra "C" e saia para tomar um café. Vamos examinar os segmentos TCP que são enviados entre o cliente e o servidor. Como mostra a Figura 3.31, admitamos que os números de sequência iniciais sejam 42 e 79 para cliente e servidor, respectivamente. Lembre-se de que o número de sequência de um segmento será o número de sequência do primeiro byte do seu campo de dados. Assim, o primeiro segmento enviado do cliente terá número de sequência 42; o primeiro segmento enviado do servidor terá número de sequência 79. Note que o número de reconhecimento será o número de sequência do próximo byte de dados que o hospedeiro estará aguardando. Após o estabelecimento da conexão TCP, mas antes de quaisquer dados serem enviados, o cliente ficará esperando pelo byte 79 e o servidor, pelo byte 42.

Como ilustra a Figura 3.31, são enviados três segmentos. O primeiro é enviado do cliente ao servidor, contendo, em seu campo de dados, um byte com a representação ASCII para a letra "C". O primeiro segmento também tem 42 em seu campo de número de sequência, como acabamos de descrever. E mais, como o cliente

FIGURA 3.31 NÚMEROS DE SEQUÊNCIA E DE RECONHECIMENTO PARA UMA APLICAÇÃO TELNET SIMPLES SOBRE TCP

ainda não recebeu nenhum dado do servidor, esse segmento terá o número 79 em seu campo de número de reconhecimento.

O segundo segmento é enviado do servidor ao cliente. Esse segmento tem dupla finalidade. A primeira é fornecer um reconhecimento para os dados que o servidor recebeu. Ao colocar 43 no campo de reconhecimento, o servidor está dizendo ao cliente que recebeu com sucesso tudo até o byte 42 e agora está aguardando os bytes de 43 em diante. A segunda finalidade desse segmento é ecoar a letra "C". Assim, o segundo segmento tem a representação ASCII de "C" em seu campo de dados. Ele tem o número de sequência 79, que é o número de sequência inicial do fluxo de dados de servidor para cliente dessa conexão TCP, pois este é o primeiríssimo byte de dados que o servidor está enviando. Note que o reconhecimento para dados do cliente para o servidor é levado em um segmento que carrega dados do servidor para o cliente. Dizemos que esse reconhecimento **pegou uma carona** (*piggybacked*) no segmento de dados do servidor ao cliente.

O terceiro é enviado do cliente ao servidor. Seu único propósito é reconhecer os dados que recebeu do servidor. (Lembre-se de que o segundo segmento continha dados — a letra "C" — do servidor para o cliente.) Ele tem um campo de dados vazio (isto é, o reconhecimento não está pegando carona com nenhum dado do cliente para o servidor). O segmento tem o número 80 no campo do número de reconhecimento porque o cliente recebeu a cadeia de dados até o byte com número de sequência 79 e agora está aguardando os bytes de 80 em diante. É possível que você esteja pensando que é estranho que esse segmento também tenha um número de sequência, já que não contém dados. Mas, como o TCP tem um campo de número de sequência, o segmento precisa apresentar algum número para preenchê-lo.

3.5.3 Estimativa do tempo de viagem de ida e volta e de esgotamento de temporização

O TCP, assim como o nosso protocolo `rdt` da Seção 3.4, utiliza um mecanismo de controle de temporização/retransmissão para recuperar segmentos perdidos. Embora conceitualmente simples, surgem muitas questões sutis quando executamos um mecanismo de controle de temporização/retransmissão em um protocolo real como o TCP. Talvez a pergunta mais óbvia seja a duração dos intervalos de controle. Claro, esse intervalo deve ser maior do que o tempo de viagem de ida e volta da conexão (RTT), isto é, o tempo decorrido entre o envio de um segmento e seu reconhecimento. Se não fosse assim, seriam enviadas retransmissões desnecessárias. Mas quão maior deve ser o intervalo e, antes de tudo, como o RTT deve ser estimado? Deve-se associar um temporizador a cada segmento não reconhecido? São tantas perguntas! Nesta seção, nossa discussão se baseia no trabalho de Jacobson [1988] sobre TCP e nas recomendações da IETF vigentes para o gerenciamento de temporizadores TCP [RFC 6298].

Estimativa do tempo de viagem de ida e volta

Vamos iniciar nosso estudo do gerenciamento do temporizador TCP considerando como esse protocolo estima o tempo de viagem de ida e volta entre remetente e destinatário, o que apresentaremos a seguir. O RTT para um segmento, denominado `SampleRTT` no exemplo, é o tempo transcorrido entre o momento em que o segmento é enviado (isto é, passado ao IP) e o momento em que é recebido um reconhecimento para ele. Em vez de medir um `SampleRTT` para cada segmento transmitido, a maioria das implementações de TCP executa apenas uma medição de `SampleRTT` por vez. Isto é, em qualquer instante, o `SampleRTT` estará sendo estimado para apenas um dos segmentos transmitidos mas ainda não reconhecidos, o que resulta em um novo valor de `SampleRTT` para mais ou menos cada RTT. E mais, o TCP nunca computa um `SampleRTT` para um segmento que foi retransmitido; apenas mede-o para segmentos que foram transmitidos uma vez [Karn, 1987]. (Um dos problemas ao final do capítulo perguntará por quê.)

Claro, os valores de `SampleRTT` sofrerão variação de segmento para segmento em decorrência do congestionamento nos roteadores e das variações de carga nos sistemas finais. Por causa dessa variação, qualquer dado valor de `SampleRTT` pode ser atípico. Portanto, para estimar um RTT comum, é natural tomar alguma espécie

de média dos valores de SampleRTT. O TCP mantém uma média, denominada EstimatedRTT, dos valores de SampleRTT. Ao obter um novo SampleRTT, o TCP atualiza EstimatedRTT de acordo com a seguinte fórmula:

EstimatedRTT = (1 – α) • EstimatedRTT + α • SampleRTT

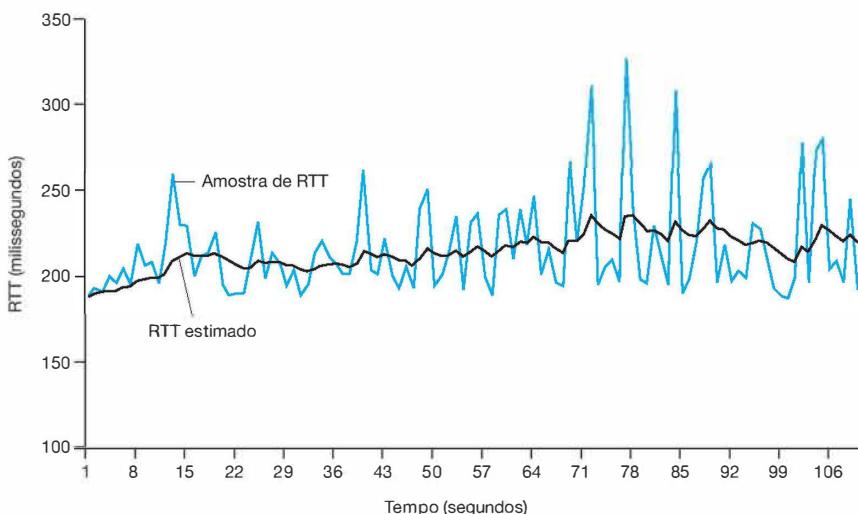
Essa fórmula está escrita na forma de um comando de linguagem de programação — o novo valor de EstimatedRTT é uma combinação ponderada entre o valor anterior de EstimatedRTT e o novo valor para SampleRTT. O valor recomendado de α é 0,125 (isto é, 1/8) [RFC 6298], caso em que essa fórmula se torna:

EstimatedRTT = 0,875 • EstimatedRTT + 0,125 • SampleRTT

Note que EstimatedRTT é uma média ponderada dos valores de SampleRTT. Como veremos em um exercício ao final deste capítulo, essa média ponderada atribui um peso maior às amostras recentes do que às antigas. Isso é natural, pois as amostras mais recentes refletem melhor o estado atual de congestionamento da rede. Em estatística, esse tipo de média é denominado **média móvel exponencial ponderada**. A palavra "exponencial" aparece na MMEP porque o peso atribuído a um dado SampleRTT diminui exponencialmente à medida que as atualizações são realizadas. Os exercícios pedirão que você derive o termo exponencial em EstimatedRTT.

A Figura 3.32 mostra os valores de SampleRTT e EstimatedRTT para um valor de α = 1/8, para uma conexão TCP entre gaia.cs.umass.edu (em Amherst, Massachusetts) e fantasia.eurecom.fr (no sul da França). Fica claro que as variações em SampleRTT são atenuadas no cálculo de EstimatedRTT.

Além de ter uma estimativa do RTT, também é valioso ter uma medida de sua variabilidade. O [RFC 6298] define a variação do RTT, DevRTT, como uma estimativa do desvio típico entre SampleRTT e EstimatedRTT:

DevRTT = (1 – β) • DevRTT + β • |SampleRTT – EstimatedRTT|

Note que DevRTT é uma MMEP da diferença entre SampleRTT e EstimatedRTT. Se os valores de SampleRTT apresentarem pouca variação, então DevRTT será pequeno; por outro lado, se houver muita variação, DevRTT será grande. O valor recomendado para β é 0,25.

PRINCÍPIOS NA PRÁTICA

O TCP fornece transferência confiável de dados usando reconhecimentos positivos e temporizadores, de modo muito parecido com o que estudamos na Seção 3.4. O protocolo reconhece dados que foram recebidos corretamente e retransmite segmentos quando entende que eles ou seus reconhecimentos correspondentes foram perdidos ou corrompidos. Certas versões do TCP também têm um mecanismo NAK implícito — com o mecanismo de retransmissão rápida do TCP. O recebimento de três ACKs duplicados para um dado segmento serve como um NAK implícito para o seguinte, acionando a retransmissão daquele segmento antes que o tempo se esgote. O TCP usa sequência de números para permitir que o destinatário identifique segmentos perdidos ou duplicados. Exatamente como no caso de nosso protocolo de transferência confiável de dados rdt3.0, o TCP em si não pode determinar com certeza se um segmento, ou seu ACK, está perdido, corrompido ou atrasado demais. No remetente, a resposta do TCP será a mesma: retransmitir o segmento.

O TCP também utiliza paralelismo, permitindo que o remetente tenha, a qualquer tempo, múltiplos segmentos transmitidos mas ainda não reconhecidos. Vimos antes que o paralelismo pode melhorar muito a vazão de uma sessão quando a razão entre o tempo de transmissão do segmento e o atraso de viagem de ida e volta é pequena. O número específico de segmentos não reconhecidos que um remetente pode ter é determinado pelos mecanismos de controle de fluxo e controle de congestionamento do TCP. O controle de fluxo do TCP é discutido no final desta seção; o controle de congestionamento do TCP é discutido na Seção 3.7. Por enquanto, devemos apenas ficar cientes de que o TCP remetente usa paralelismo.

FIGURA 3.32 AMOSTRAS E ESTIMATIVAS DE RTT

Estabelecimento e gerenciamento da temporização de retransmissão

Dados valores de `EstimatedRTT` e `DevRTT`, que valor deve ser utilizado para a temporização de retransmissão do TCP? É óbvio que o intervalo deve ser maior ou igual a `EstimatedRTT`, caso contrário seriam enviadas retransmissões desnecessárias. Mas a temporização de retransmissão não deve ser muito maior do que `EstimatedRTT`, senão, quando um segmento fosse perdido, o TCP não o retransmitiria rápido, o que resultaria em grandes atrasos de transferência de dados. Portanto, é desejável que o valor estabelecido para a temporização seja igual a `EstimatedRTT` mais certa margem, que deverá ser grande quando houver muita variação nos valores de `SampleRTT` e pequena quando houver pouca variação. Assim, o valor de `DevRTT` deverá entrar em cena. Todas essas considerações são levadas em conta no método do TCP para determinar a temporização de retransmissão:

```
TimeoutInterval = EstimatedRTT + 4 • DevRTT
```

Recomenda-se um valor inicial de 1 segundo para `TimeoutInterval` [RFC 6298]. Além disso, quando há temporização, o valor de `TimeoutInterval` é dobrado para evitar que haja uma temporização para um segmento subsequente, que logo será reconhecido. Porém, assim que o segmento é recebido e `EstimatedRTT` é atualizado, o `TimeoutInterval` novamente é calculado usando a fórmula anterior.

3.5.4 Transferência confiável de dados

Lembre-se de que o serviço da camada de rede da Internet (serviço IP) não é confiável. O IP não garante a entrega de datagramas na ordem correta nem a integridade de seus dados nos datagramas. Com o serviço IP, os datagramas podem transbordar dos buffers dos roteadores e jamais alcançar seu destino; podem também chegar fora de ordem. Além disso, os bits dos datagramas podem ser corrompidos (passar de 0 para 1 e vice-versa). Como os segmentos da camada de transporte são carregados pela rede por datagramas IPs, também eles podem sofrer esses mesmos problemas.

O TCP cria um **serviço de transferência confiável de dados** sobre o serviço de melhor esforço do IP. Esse serviço de transferência garante que a cadeia de dados que um processo lê a partir de seu buffer de recebimento TCP não está corrompida, não tem lacunas, não tem duplicações e está em sequência, isto é, a cadeia de bytes

é idêntica à cadeia de bytes enviada pelo sistema final que está do outro lado da conexão. O modo como o TCP oferece transferência confiável de dados envolve muitos dos princípios estudados na Seção 3.4.

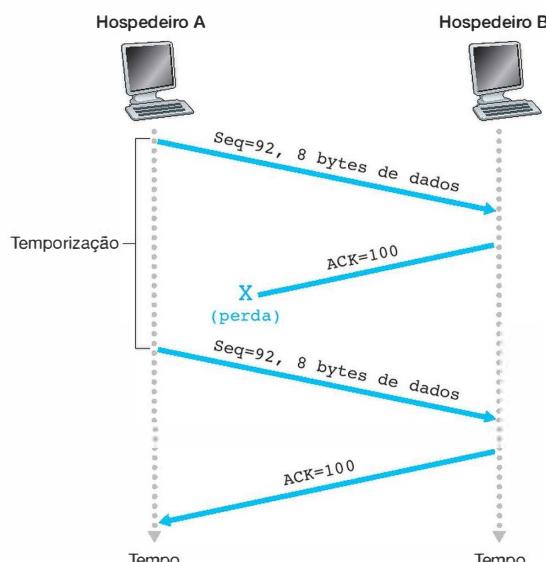
Quando desenvolvemos técnicas de transferência confiável de dados, era conceitualmente mais fácil admitir que existia um temporizador individual associado com cada segmento transmitido mas ainda não reconhecido. Embora, em teoria, isso seja ótimo, o gerenciamento de temporizadores pode exigir considerável sobrecarga. Assim, os procedimentos recomendados no [RFC 6298] para gerenciamento de temporizadores TCP utilizam apenas um *único* temporizador de retransmissão, mesmo que haja vários segmentos transmitidos ainda não reconhecidos. O protocolo TCP apresentado nesta seção segue essa recomendação.

Discutiremos como o TCP provê transferência confiável de dados em duas etapas incrementais. Primeiro, apresentamos uma descrição muito simplificada de um remetente TCP que utiliza apenas controle de temporizadores para se recuperar da perda de segmentos; em seguida, apresentaremos uma descrição mais complexa, que utiliza reconhecimentos duplicados além de temporizadores de retransmissão. Na discussão que se segue, admitimos que os dados estão sendo enviados em uma direção somente, do hospedeiro A ao hospedeiro B, e que o hospedeiro A está enviando um arquivo grande.

A Figura 3.33 apresenta uma descrição muito simplificada de um remetente TCP. Vemos que há três eventos importantes relacionados com a transmissão e a retransmissão de dados no TCP remetente: dados recebidos da

FIGURA 3.33 REMETENTE TCP SIMPLIFICADO

```
/* Suponha que o remetente não seja compelido pelo fluxo de TCP ou controle
de congestionamento, que o tamanho dos dados vindos de cima seja menor do que
o MSS e que a transferência de dados ocorra apenas em uma direção. */

NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumber

loop (forever) { switch(event)
      event: dados recebidos da aplicação de cima
            criar segmento TCP com número de sequência NextSeqNum
            if (temporizador atualmente parado)
                  iniciar temporizador
            passar segmento ao IP
            NextSeqNum=NextSeqNum+length(dados)
            break;

      event: esgotamento do temporizador
            retransmitir segmento ainda não reconhecido com o menor
                  número de sequência
            iniciar temporizador
            break;

      event: ACK recebido, com valor do campo ACK de y
            if (y > SendBase) {
                  SendBase=y
                  if (não há atualmente segmentos ainda não reconhecidos)
                        iniciar temporizador
                  }
            break;

} /* fim do loop forever */
```

aplicação; esgotamento do temporizador e recebimento de ACK. Quando ocorre o primeiro evento importante, o TCP recebe dados da camada de aplicação, encapsula-os em um segmento e passa-o ao IP. Note que cada segmento inclui um número de sequência que é o número da corrente de bytes do primeiro byte de dados no segmento, como descrito na Seção 3.5.2. Note também que, se o temporizador não estiver funcionando naquele instante para algum outro segmento, o TCP aciona o temporizador quando o segmento é passado para o IP. (Fica mais fácil se você imaginar que o temporizador está associado com o mais antigo segmento não reconhecido.) O intervalo de expiração para esse temporizador é o TimeoutInterval, calculado a partir de `EstimatedRTT` e `DevRTT`, como descrito na Seção 3.5.3.

O segundo é o esgotamento do temporizador. O TCP responde a esse evento retransmitindo o segmento que causou o esgotamento da temporização e então reinicia o temporizador.

O terceiro evento importante que deve ser manipulado pelo TCP remetente é a chegada de um segmento de reconhecimento (ACK) do destinatário (mais especificamente, um segmento contendo um valor de campo de ACK válido). Quando da ocorrência, o TCP compara o valor do ACK, y, com sua variável `SendBase`. A variável de estado SendBase do TCP é o número de sequência do mais antigo byte não reconhecido. (Assim, `SendBase–1` é o número de sequência do último byte que se sabe ter sido recebido pelo destinatário de modo correto e na ordem certa.) Como comentamos, o TCP usa reconhecimentos cumulativos, de maneira que y reconhece o recebimento de todos os bytes antes do byte número `y`. Se `y > SendBase`, então o ACK está reconhecendo um ou mais bytes não reconhecidos antes. Desse modo, o remetente atualiza sua variável `SendBase` e também reinicia o temporizador se houver quaisquer segmentos ainda não reconhecidos.

Alguns cenários interessantes

Acabamos de descrever uma versão muito simplificada do modo como o TCP provê transferência confiável de dados, mas mesmo essa descrição tão simplificada tem muitas sutilezas. Para ter uma boa ideia de como esse protocolo funciona, vamos agora examinar alguns cenários simples. A Figura 3.34 ilustra o primeiro cenário, em que um hospedeiro A envia um segmento ao hospedeiro B. Suponha que esse segmento tenha número de sequência

FIGURA 3.34 RETRANSMISSÃO DEVIDO A UM RECONHECIMENTO PERDIDO

92 e contenha 8 bytes de dados. Após enviá-lo, o hospedeiro A espera por um segmento de B com número de reconhecimento 100. Embora o segmento de A seja recebido em B, o reconhecimento de B para A se perde. Nesse caso, ocorre o evento de expiração do temporizador e o hospedeiro A retransmite o mesmo segmento. É claro que, quando recebe a retransmissão, o hospedeiro B observa, pelo número de sequência, que o segmento contém dados que já foram recebidos. Assim, o TCP no hospedeiro B descarta os bytes do segmento retransmitido.

Em um segundo cenário, mostrado na Figura 3.35, o hospedeiro A envia dois segmentos seguidos. O primeiro tem número de sequência 92 e 8 bytes de dados. O segundo tem número de sequência 100 e 20 bytes de dados. Suponha que ambos cheguem intactos em B e que B envie dois reconhecimentos separados para cada um desses segmentos. O primeiro deles tem número de reconhecimento 100; o segundo, número 120. Suponha agora que nenhum dos reconhecimentos chegue ao hospedeiro A antes do esgotamento do temporizador. Quando ocorre o evento de expiração do temporizador, o hospedeiro A reenvia o primeiro segmento com número de sequência 92 e reinicia o temporizador. Contanto que o ACK do segundo segmento chegue antes que o temporizador expire novamente, o segundo segmento não será retransmitido.

Em um terceiro e último cenário, suponha que o hospedeiro A envie dois segmentos, assim como no segundo exemplo. O reconhecimento do primeiro segmento é perdido na rede, mas, um pouco antes do evento de expiração, A recebe um reconhecimento com número 120. O hospedeiro A, portanto, sabe que B recebeu *tudo* até o byte 119; logo, ele não reenvia nenhum dos dois segmentos. Tal cenário está ilustrado na Figura 3.36.

Duplicação do tempo de expiração

Discutiremos agora algumas modificações empregadas por grande parte das execuções do TCP. A primeira refere-se à duração do tempo de expiração após a expiração de um temporizador. Nessa modificação, sempre que ocorre tal evento, o TCP retransmite o segmento ainda não reconhecido que tenha o menor número de sequência, como descrevemos anteriormente. Mas, a cada retransmissão, o TCP ajusta o próximo tempo de expiração para o dobro do valor anterior em vez de derivá-lo dos últimos `Estimated-RTT` e `DevRTT` (como descrito na

FIGURA 3.35 SEGMENTO 100 NÃO RETRANSMITIDO

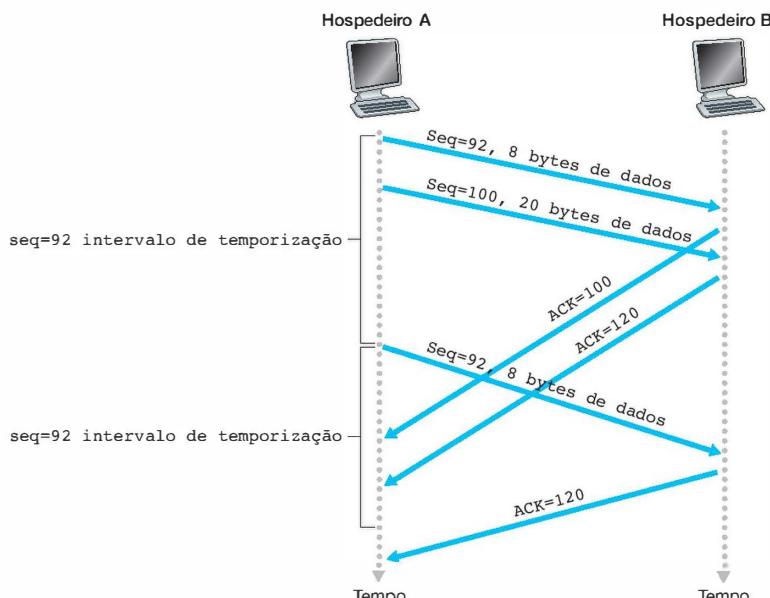
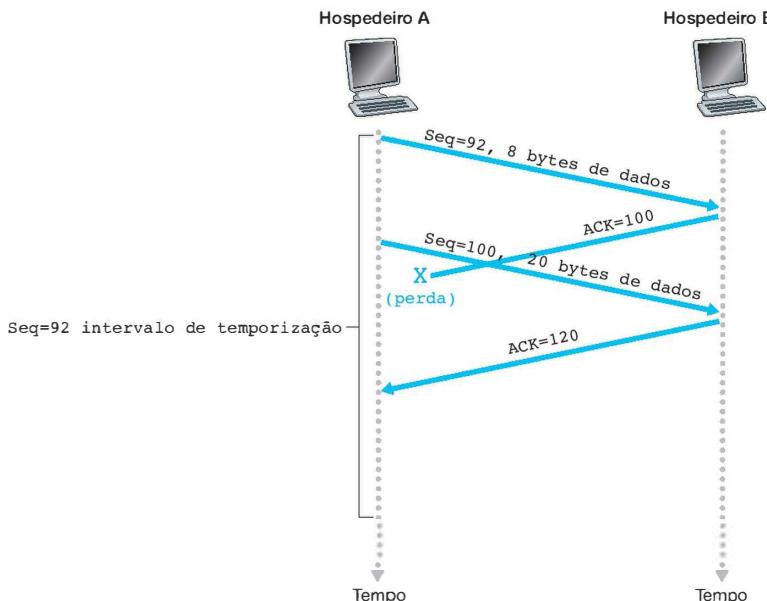



FIGURA 3.36 UM RECONHECIMENTO CUMULATIVO EVITA RETRANSMISSÃO DO PRIMEIRO SEGMENTO

Seção 3.5.3). Por exemplo, suponha que o `TimeoutInterval` associado com o mais antigo segmento ainda não reconhecido seja 0,75 s quando o temporizador expirar pela primeira vez. O TCP então retransmite esse segmento e ajusta o novo tempo de expiração para 1,5 s. Se o temporizador expirar novamente 1,5 s mais tarde, o TCP retransmitirá de novo esse segmento, agora ajustando o tempo de expiração para 3,0 s. Assim, o tempo aumenta exponencialmente após cada retransmissão. Todavia, sempre que o temporizador é iniciado após qualquer um dos outros dois eventos (isto é, dados recebidos da aplicação anterior e ACK recebido), o `TimeoutInterval` será derivado dos valores mais recentes de `EstimatedRTT` e `DevRTT`.

Essa modificação provê uma forma limitada de controle de congestionamento. (Maneiras mais abrangentes de controle de congestionamento no TCP serão estudadas na Seção 3.7.) A causa mais provável da expiração do temporizador é o congestionamento na rede, isto é, um número muito grande de pacotes chegando a uma (ou mais) fila de roteadores no caminho entre a origem e o destino, o que provoca descarte de pacotes e/ou longos atrasos de fila. Se as origens continuarem a retransmitir pacotes de modo persistente durante um congestionamento, ele pode piorar. Em vez disso, o TCP age com mais educação: cada remetente retransmite após intervalos cada vez mais longos. Veremos que uma ideia semelhante a essa é utilizada pela Ethernet, quando estudarmos CSMA/CD no Capítulo 5.

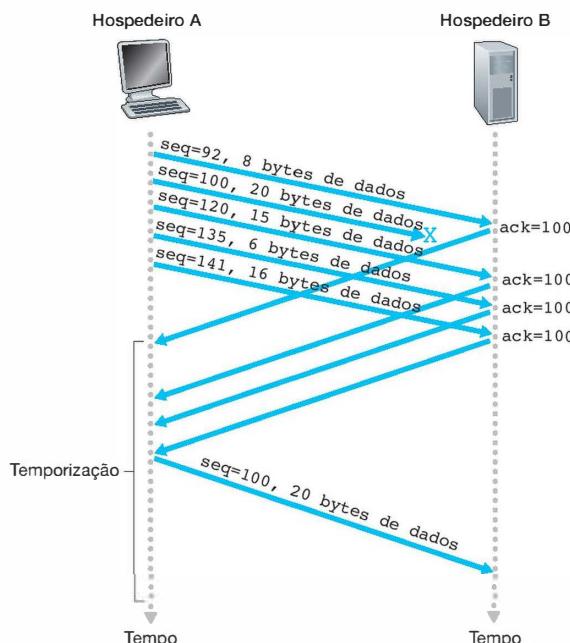
Retransmissão rápida

Um dos problemas de retransmissões acionadas por expiração de temporizador é que o período de expiração pode ser um tanto longo. Quando um segmento é perdido, esse longo período força o remetente a atrasar o reenvio do pacote perdido, aumentando por conseguinte o atraso fim a fim. Felizmente, o remetente pode com frequência detectar perda de pacote bem antes de ocorrer o evento de expiração, observando os denominados ACKs duplicados. Um **ACK duplicado** é um ACK que reconhece novamente um segmento para o qual o remetente já recebeu um reconhecimento anterior. Para entender a resposta do remetente a um ACK duplicado, devemos examinar por que o destinatário envia um ACK duplicado em primeiro lugar. A Tabela 3.2 resume a política de geração de ACKs do TCP destinatário [RFC 5681]. Quando um TCP destinatário recebe um segmento com

número de sequência maior do que o seguinte, esperado, na ordem, ele detecta uma lacuna no fluxo de dados — ou seja, um segmento faltando. Essa lacuna poderia ser o resultado de segmentos perdidos ou reordenados dentro da rede. Como o TCP não usa reconhecimentos negativos, o destinatário não pode enviar um reconhecimento negativo explícito de volta ao destinatário. Em vez disso, ele apenas reconhece mais uma vez (ou seja, gera um ACK duplicado para) o último byte de dados na ordem que foi recebido. (Observe que a Tabela 3.2 tem provisão para o caso em que o destinatário não descarta segmentos fora de ordem.)

TABELA 3.2 TCP ACK GENERATION RECOMMENDATION [RFC 5681]

Evento	Ação do TCP destinatário
Chegada de segmento na ordem com número de sequência esperado. Todos os dados até o número de sequência esperado já reconhecidos.	ACK retardado. Espera de até 500 ms pela chegada de outro segmento na ordem. Se o segmento seguinte na ordem não chegar nesse intervalo, envia um ACK.
Chegada de segmento na ordem com número de sequência esperado. Outro segmento na ordem esperando por transmissão de ACK.	Envio imediato de um único ACK cumulativo, reconhecendo ambos os segmentos.
Chegada de um segmento fora da ordem com número de sequência mais alto do que o esperado. Lacuna detectada.	Envio imediato de um ACK duplicado, indicando número de sequência do byte seguinte esperado (que é a extremidade mais baixa da lacuna).
Chegada de um segmento que preenche, parcial ou completamente, a lacuna nos dados recebidos.	Envio imediato de um ACK, contanto que o segmento comece na extremidade mais baixa da lacuna.

Como um remetente quase sempre envia um grande número de segmentos, um atrás do outro, se um segmento for perdido, provavelmente existirão muitos ACKs duplicados, também um após o outro. Se o TCP remetente receber três ACKs duplicados para os mesmos dados, ele tomará isso como indicação de que o segmento que se seguiu ao segmento reconhecido três vezes foi perdido. (Nos exercícios de fixação consideraremos por que o remetente espera três ACKs duplicados e não apenas um.) No caso de receber três ACKs duplicados, o TCP remetente realiza uma **retransmissão rápida** [RFC 5681], retransmitindo o segmento que falta *antes* da expiração do temporizador do segmento. Isso é mostrado na Figura 3.37, em que o segundo segmento é perdido, e então retransmitido antes da expiração do temporizador. Para o TCP com retransmissão rápida, o seguinte trecho de codificação substitui o evento ACK recebido na Figura 3.33:

```
evento: ACK recebido, com valor do campo ACK y
      if (y > SendBase) {
            SendBase=y
            if (atualmente ainda não há segmentos reconhecidos)
                  iniciar temporizador
            }
      else { /* um ACK duplicado para segmento já reconhecido */
            incrementar número de ACKs duplicados recebidos para y
            if (número de ACKS duplicados recebidos para y==3)
                  /* retransmissão rápida do TCP */
                  reenviar segmento com número de sequência y
            }
      break;
```

Observamos antes que muitas questões sutis vêm à tona quando um mecanismo de controle de temporização/retransmissão é executado em um protocolo real como o TCP. Os procedimentos anteriores, cuja evolução é resultado de mais de 15 anos de experiência com temporizadores TCP, devem convencê-lo de que, na realidade, é isso que acontece!

FIGURA 3.37 RETRANSMISSÃO RÁPIDA: RETRANSMITIR O SEGMENTO QUE FALTA ANTES DA EXPIRAÇÃO DO TEMPORIZADOR DO SEGMENTO

Go-Back-N ou repetição seletiva?

Vamos encerrar nosso estudo do mecanismo de recuperação de erros do TCP considerando a seguinte pergunta: o TCP é um protocolo GBN ou SR? Lembre-se de que, no TCP, os reconhecimentos são cumulativos e segmentos recebidos de modo correto, mas fora da ordem, não são reconhecidos (ACK) individualmente pelo destinatário. Em consequência, como mostrou a Figura 3.33 (veja também a Figura 3.19), o TCP remetente precisa tão somente lembrar o menor número de sequência de um byte transmitido, porém não reconhecido (`SendBase`) e o número de sequência do byte seguinte a ser enviado (`NextSeqNum`). Nesse sentido, o TCP se parece muito com um protocolo ao estilo do GBN. Porém, há algumas diferenças surpreendentes entre o TCP e o GBN. Muitas execuções do TCP armazenarão segmentos recebidos corretamente, mas fora da ordem [Stevens, 1994]. Considere também o que acontece quando o remetente envia uma sequência de segmentos 1, 2, ..., N e todos os segmentos chegam ao destinatário na ordem e sem erro. Além disso, suponha que o reconhecimento para o pacote $n < N$ se perca, mas que os $N - 1$ reconhecimentos restantes cheguem ao remetente antes do esgotamento de suas respectivas temporizações. Nesse exemplo, o GBN retransmitiria não só o pacote n, mas também todos os subsequentes $n + 1$, $n + 2$, ..., N. O TCP, por outro lado, retransmitiria no máximo um segmento, a saber, n. E mais, o TCP nem ao menos retransmitiria o segmento n se o reconhecimento para $n + 1$ chegasse antes do final da temporização para o segmento n.

Uma modificação proposta para o TCP, denominada **reconhecimento seletivo** [RFC 2018], permite que um destinatário TCP reconheça seletivamente segmentos fora de ordem, em vez de apenas reconhecer de modo cumulativo o último segmento recebido corretamente e na ordem. Quando combinado com retransmissão seletiva — isto é, saltar a retransmissão de segmentos que já foram reconhecidos de modo seletivo pelo destinatário —, o TCP se parece muito com nosso protocolo SR genérico. Assim, o mecanismo de recuperação de erros do TCP talvez seja mais bem caracterizado como um híbrido dos protocolos GBN e SR.

3.5.5 Controle de fluxo

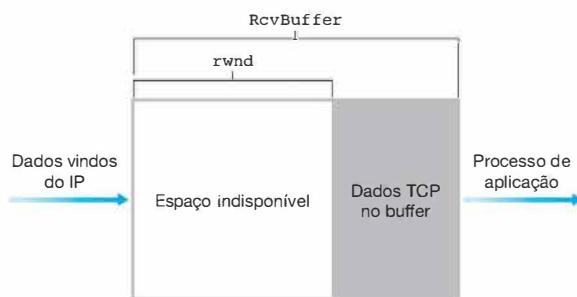
Lembre-se de que os hospedeiros de cada lado de uma conexão TCP reservam um buffer de recepção para a conexão. Quando a conexão TCP recebe bytes que estão corretos e em sequência, ele coloca os dados no buffer de recepção. O processo de aplicação associado lerá os dados a partir desse buffer, mas não necessariamente no momento em que são recebidos. Na verdade, a aplicação receptora pode estar ocupada com alguma outra tarefa e nem ao menos tentar ler os dados até muito após a chegada deles. Se a aplicação for relativamente lenta na leitura dos dados, o remetente pode muito facilmente saturar o buffer de recepção da conexão por enviar demasiados dados muito rapidamente.

O TCP provê um **serviço de controle de fluxo** às suas aplicações, para eliminar a possibilidade de o remetente estourar o buffer do destinatário. Assim, controle de fluxo é um serviço de compatibilização de velocidades — compatibiliza a taxa à qual o remetente está enviando com aquela à qual a aplicação receptora está lendo. Como já notamos, um TCP remetente também pode ser estrangulado por causa do congestionamento dentro da rede IP. Esse modo de controle do remetente é denominado **controle de congestionamento**, um tópico que será examinado em detalhes nas Seções 3.6 e 3.7. Mesmo que as ações executadas pelo controle de fluxo e pelo controle de congestionamento sejam semelhantes (a regulagem do remetente), fica evidente que elas são executadas por razões muito diferentes. Infelizmente, muitos autores usam os termos de modo intercambiável, e o leitor esperto tem de tomar muito cuidado para distinguir os dois casos. Vamos agora discutir como o TCP provê seu serviço de controle de fluxo. Para podermos enxergar o quadro geral, sem nos fixarmos nos detalhes, nesta seção admitiremos que essa implementação do TCP é tal que o receptor TCP descarta segmentos fora da ordem.

O TCP oferece serviço de controle de fluxo fazendo que o *remetente* mantenha uma variável denominada **janela de recepção**. De modo informal, a janela de recepção é usada para dar ao remetente uma ideia do espaço de buffer livre disponível no destinatário. Como o TCP é *full-duplex*, o remetente de cada lado da conexão mantém uma janela de recepção distinta. Vamos examinar a janela de recepção no contexto de uma transferência de arquivo. Suponha que o hospedeiro A esteja enviando um arquivo grande ao hospedeiro B por uma conexão TCP. O hospedeiro B aloca um buffer de recepção a essa conexão; denominemos seu tamanho `RcvBuffer`. De tempos em tempos, o processo de aplicação no hospedeiro B faz a leitura do buffer. São definidas as seguintes variáveis:

- `LastByteRead`: número do último byte na cadeia de dados lido do buffer pelo processo de aplicação em B.
- `LastByteRcvd`: número do último byte na cadeia de dados que chegou da rede e foi colocado no buffer de recepção de B.

Como o TCP não tem permissão para saturar o buffer alocado, devemos ter:

```
LastByteRcvd – LastByteRead ≤ RcvBuffer
```

A janela de recepção, denominada `rwnd`, é ajustada para a quantidade de espaço disponível no buffer:

```
rwnd = RcvBuffer – [LastByteRcvd – LastByteRead]
```

Como o espaço disponível muda com o tempo, `rwnd` é dinâmica. Esta variável está ilustrada na Figura 3.38.

Como a conexão usa a variável `rwnd` para prover o serviço de controle de fluxo? O hospedeiro B diz ao hospedeiro A quanto espaço disponível ele tem no buffer da conexão colocando o valor corrente de `rwnd` no campo de janela de recepção de cada segmento que envia a A. No começo, o hospedeiro B estabelece `rwnd = RcvBuffer`. Note que, para conseguir isso, o hospedeiro B deve monitorar diversas variáveis específicas da conexão.

O hospedeiro A, por sua vez, monitora duas variáveis, `LastByteSent` e `LastByteAcked`, cujos significados são óbvios. Note que a diferença entre essas duas variáveis, `LastByteSent — LastByteAcked`, é a quantidade de dados não reconhecidos que A enviou para a conexão. Mantendo a quantidade de dados não

FIGURA 3.38 A JANELA DE RECEPÇÃO (`rwnd`) E O BUFFER DE RECEPÇÃO (`RcvBuffer`)

reconhecidos menor que o valor de `rwnd`, o hospedeiro A tem certeza de que não está fazendo transbordar o buffer de recepção no hospedeiro B. Assim, A tem de certificar-se, durante toda a duração da conexão, de que:

```
LastByteSent – LastByteAcked ≤ rwnd
```

Há um pequeno problema técnico com esse esquema. Para percebê-lo, suponha que o buffer de recepção do hospedeiro B fique tão cheio que `rwnd` = 0. Após anunciar ao hospedeiro A que `rwnd` = 0, imagine que B não tenha *nada* para enviar ao hospedeiro A. Agora considere o que acontece. Enquanto o processo de aplicação em B esvazia o buffer, o TCP não envia novos segmentos com novos valores `rwnd` para o hospedeiro A. Na verdade, o TCP lhe enviará um segmento somente se tiver dados ou um reconhecimento para enviar. Por conseguinte, o hospedeiro A nunca será informado de que foi aberto algum espaço no buffer de recepção do hospedeiro B: ele ficará bloqueado e não poderá transmitir mais dados! Para resolver esse problema, a especificação do TCP requer que o hospedeiro A continue a enviar segmentos com um byte de dados quando a janela de recepção de B for zero. Esses segmentos serão reconhecidos pelo receptor. Por fim, o buffer começará a esvaziar, e os reconhecimentos conterão um valor diferente de zero em `rwnd`.

O site de apoio deste livro fornece um applet interativo em Java que ilustra a operação da janela de recepção do TCP.

Agora que descrevemos o serviço de controle de fluxo do TCP, mencionaremos de maneira breve que o UDP não provê controle de fluxo. Para entender a questão, considere o envio de uma série de segmentos UDP de um processo no hospedeiro A para um processo no hospedeiro B. Para uma execução UDP típica, o UDP anexará os segmentos a um buffer de tamanho finito que "precede" o *socket* correspondente (isto é, o *socket* para o processo). O processo lê um segmento inteiro do buffer por vez. Se o processo não ler os segmentos com rapidez suficiente, o buffer transbordará e os segmentos serão descartados.

3.5.6 Gerenciamento da conexão TCP

Nesta subseção, examinamos mais de perto como uma conexão TCP é estabelecida e encerrada. Embora esse tópico talvez não pareça de particular interesse, é importante, porque o estabelecimento da conexão TCP tem um peso significativo nos atrasos percebidos (por exemplo, ao navegar pela Web). Além disso, muitos dos ataques mais comuns a redes — entre eles o incrivelmente popular ataque de inundação SYN — exploram vulnerabilidades no gerenciamento da conexão TCP. Em primeiro lugar, vamos ver como essa conexão é estabelecida. Suponha que um processo que roda em um hospedeiro (cliente) queira iniciar uma conexão com outro processo em outro hospedeiro (servidor). O processo de aplicação cliente primeiro informa ao TCP cliente que quer estabelecer uma conexão com um processo no servidor. O TCP no cliente então estabelece uma conexão TCP com o TCP no servidor da seguinte maneira:

- *Etapa 1.* O lado cliente do TCP primeiro envia um segmento TCP especial ao lado servidor do TCP. Esse segmento não contém nenhum dado de camada de aplicação, mas um dos bits de *flag* no seu cabeçalho

(veja a Figura 3.29), o bit SYN, é ajustado para 1. Por essa razão, o segmento é denominado um segmento SYN. Além disso, o cliente escolhe ao acaso um número de sequência inicial (`client_isn`) e o coloca no campo de número de sequência do segmento TCP SYN inicial. Esse segmento é encapsulado em um datagrama IP e enviado ao servidor. A aleatoriedade adequada da escolha de `client_isn` de modo a evitar certos ataques à segurança tem despertado considerável interesse [CERT 2001-09].

- *Etapa 2.* Assim que o datagrama IP contendo o segmento TCP SYN chega ao hospedeiro servidor (admitindo-se que ele chegue mesmo!), o servidor extrai o segmento TCP SYN do datagrama, aloca buffers e variáveis TCP à conexão e envia um segmento de aceitação de conexão ao TCP cliente. (Veremos, no Capítulo 8, que a alocação desses buffers e variáveis, antes da conclusão da terceira etapa da apresentação de três vias, torna o TCP vulnerável a um ataque de recusa de serviço conhecido como inundação SYN.) Esse segmento de aceitação de conexão também não contém nenhum dado de camada de aplicação. Contudo, contém três informações importantes no cabeçalho do segmento: o bit SYN está com valor 1; o campo de reconhecimento do cabeçalho do segmento TCP está ajustado para `client_isn+1`; e, por fim, o servidor escolhe seu próprio número de sequência inicial (`server_isn`) e coloca esse valor no campo de número de sequência do cabeçalho do segmento TCP. Esse segmento de aceitação de conexão está dizendo, com efeito, "Recebi seu pacote SYN para começar uma conexão com seu número de sequência inicial client_isn. Concordo em estabelecer essa conexão. Meu número de sequência inicial é server_isn". O segmento de concessão da conexão às vezes é denominado **segmento SYNACK**.
- *Etapa 3.* Ao receber o segmento SYNACK, o cliente também reserva buffers e variáveis para a conexão. O hospedeiro cliente então envia ao servidor mais um segmento. Este último reconhece o segmento de confirmação da conexão do servidor (o cliente o faz colocando o valor `server_isn+1` no campo de reconhecimento do cabeçalho do segmento TCP). O bit SYN é ajustado para 0, já que a conexão está estabelecida. A terceira etapa da apresentação de três vias pode conduzir os dados cliente/servidor na carga útil do segmento.

Completadas as três etapas, os hospedeiros cliente e servidor podem enviar segmentos contendo dados um ao outro. Em cada um desses futuros segmentos, o bit SYN estará ajustado para 0. Note que, para estabelecer a conexão, três pacotes são enviados entre dois hospedeiros, como ilustra a Figura 3.39. Por tal razão, esse procedimento de estabelecimento de conexão é com frequência denominado **apresentação de três vias**. Vários aspectos da apresentação de três vias do TCP são tratados nos exercícios ao final deste capítulo (Por que são necessários os números de sequência iniciais? Por que é preciso uma apresentação de três vias, e não apenas de duas vias?). É interessante notar que um alpinista e seu amarrador (que fica mais abaixo e cuja tarefa é passar a corda de segurança ao alpinista) usam um protocolo de comunicação de apresentação de três vias idêntico ao do TCP para garantir que ambos os lados estejam prontos antes de o alpinista iniciar a escalada.

Tudo o que é bom dura pouco, e o mesmo é válido para uma conexão TCP. Qualquer um dos dois processos que participam de uma conexão TCP pode encerrar a conexão. Quando esta termina, os "recursos" (isto é, os buffers e as variáveis) nos hospedeiros são liberados. Como exemplo, suponha que o cliente decida encerrar a conexão, como mostra a Figura 3.40. O processo de aplicação cliente emite um comando para fechar. Isso faz que o TCP cliente envie um segmento TCP especial ao processo servidor, cujo bit de *flag* no cabeçalho do segmento, denominado bit FIN (veja a Figura 3.29), tem valor ajustado em 1. Quando o servidor recebe esse segmento, envia de volta ao cliente um segmento de reconhecimento. O servidor então envia seu próprio segmento de encerramento, que tem o bit FIN ajustado em 1. Por fim, o cliente reconhece o segmento de encerramento do servidor. Nesse ponto, todos os recursos dos dois hospedeiros estão liberados.

Durante a vida de uma conexão TCP, o protocolo TCP que roda em cada hospedeiro faz transições pelos vários **estados do TCP**. A Figura 3.41 ilustra uma sequência típica de estados do TCP visitados pelo TCP cliente. O TCP cliente começa no estado CLOSED. A aplicação no lado cliente inicia uma nova conexão TCP (criando um objeto *socket* como nos exemplos em Java do Capítulo 2). Isso faz o TCP no cliente enviar um segmento SYN ao TCP no servidor. Após o envio, o TCP cliente entra no estado SYN_SENT e, enquanto isso, o TCP cliente espera por um segmento do TCP servidor que inclui um reconhecimento para o segmento anterior do cliente, e tem o bit SYN ajustado para o valor 1. Assim que recebe esse segmento, o TCP cliente entra no estado ESTABLISHED, quando pode enviar e receber segmentos TCP que contêm carga útil de dados (isto é, gerados pela aplicação).

Suponha que a aplicação cliente decida que quer fechar a conexão. (Note que o servidor também tem a alternativa de fechá-la.) Isso faz o TCP cliente enviar um segmento TCP com o bit FIN ajustado em 1 e entrar no estado FIN_WAIT_1. No estado FIN_WAIT_1, o TCP cliente espera por um segmento TCP do servidor com um reconhecimento. Quando recebe esse segmento, o TCP cliente entra no estado FIN_WAIT_2. No estado FIN_ WAIT_2, ele espera por outro segmento do servidor com o bit FIN ajustado para 1. Após recebê-lo, o TCP

FIGURA 3.39 APRESENTAÇÃO DE TRÊS VIAS DO TCP: TROCA DE SEGMENTOS

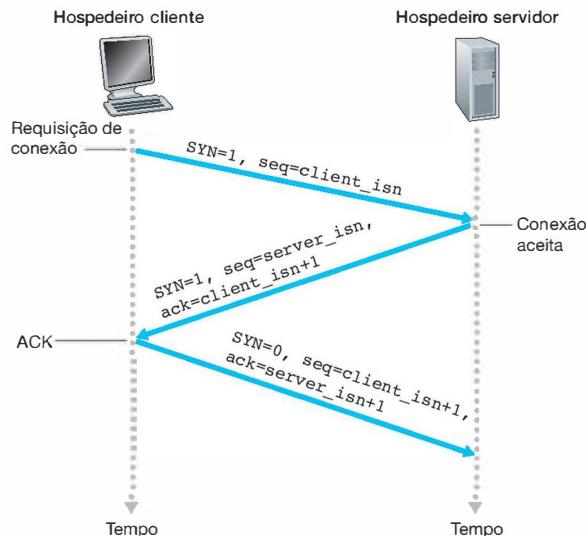



FIGURA 3.40 ENCERRAMENTO DE UMA CONEXÃO TCP

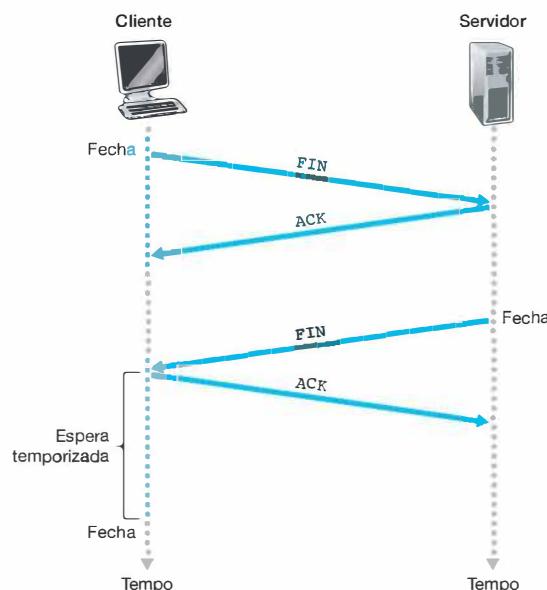
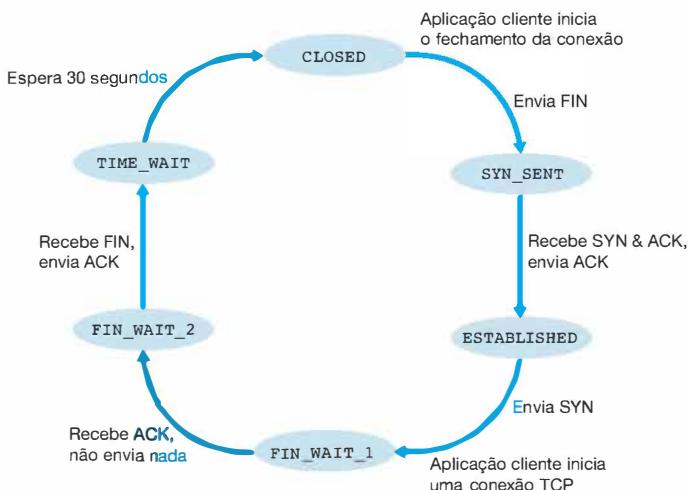
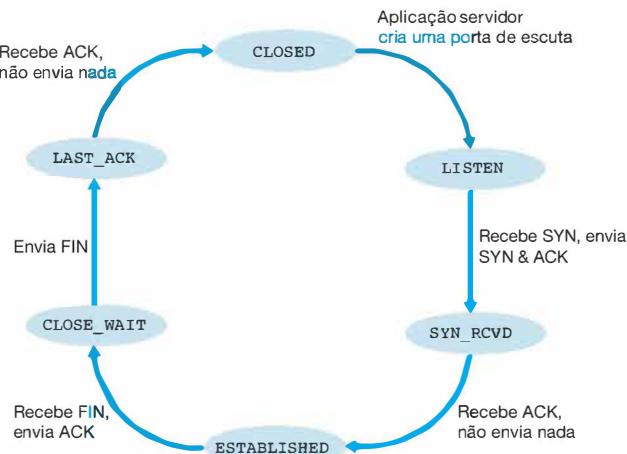



FIGURA 3.41 UMA SEQUÊNCIA TÍPICA DE ESTADOS DO TCP VISITADOS POR UM TCP CLIENTE

cliente reconhece o segmento do servidor e entra no estado TIME_WAIT. Esse estado permite que o TCP cliente reenvie o reconhecimento final, caso o ACK seja perdido. O tempo passado no estado TIME_WAIT depende da implementação, mas os valores típicos são 30 s, 1 min e 2 min. Após a espera, a conexão se encerra formalmente e todos os recursos do lado cliente (inclusive os números de porta) são liberados.

A Figura 3.42 ilustra a série de estados normalmente visitados pelo TCP do lado servidor, admitindo-se que é o cliente quem inicia o encerramento da conexão. As transições são autoexplicativas. Nesses dois diagramas de transição de estados, mostramos apenas como uma conexão TCP é em geral estabelecida e fechada. Não descrevemos o que acontece em certos cenários patológicos, por exemplo, quando ambos os lados de uma conexão querem iniciar ou fechar ao mesmo tempo. Se estiver interessado em aprender mais sobre esse assunto e sobre outros mais avançados referentes ao TCP, consulte o abrangente livro de Stevens [1994].

FIGURA 3.42 UMA SEQUÊNCIA TÍPICA DE ESTADOS DO TCP VISITADOS POR UM TCP DO LADO DO SERVIDOR

SEGURANÇA EM FOCO

O ataque Syn Flood

Vimos em nossa discussão sobre a apresentação de três vias do TCP que um servidor aloca e inicializa as variáveis da conexão e os buffers em resposta ao SYN recebido. O servidor, então, envia um SYNACK em resposta e aguarda um segmento ACK do cliente. Se o cliente não enviar um ACK para completar o terceiro passo da apresentação de três vias, com o tempo (em geral após um minuto ou mais), o servidor finalizará a conexão semiaberta e recuperará os recursos alocadas.

Esse protocolo de gerenciamento da conexão TCP abre caminho para um ataque DoS clássico, ou seja, o **ataque SYN flood**. Neste ataque, o vilão envia um grande número de segmentos SYN TCP, sem concluir a terceira etapa de apresentação. Com esse acúmulo de segmentos SYN, os recursos de conexão do servidor podem se esgotar depressa já que são alocados (mas nunca usados) para conexões semiabertas; clientes legítimos, então, não são atendidos. Esses ataques SYN flood [CERT SYN, 1996] estavam entre os primeiros ataques DoS documentados pelo CERT [CERT, 2009]. Felizmente, uma defesa eficaz, conhecida como **SYN cookies** [RFC 4987], agora é empregada na maioria dos principais sistemas operacionais. SYS *cookies* funcionam da seguinte forma:

- Quando o servidor recebe um segmento SYN, não se sabe se ele vem de um usuário verdadeiro ou se é parte desse ataque. Então, em vez de criar uma conexão TCP semiaberta para esse SYN, o servidor cria um número de sequência TCP inicial, que é uma função *hash* de endereços de origem e endereços de destino IP e números de porta do segmento SYN, assim como de um número secreto conhecido apenas pelo usuário. Esse número de sequência inicial criado cuidadosamente é o assim chamado "*cookie*". O servidor, então, envia ao cliente um pacote SYNACK com esse número de sequência especial. É importante mencionar que o servidor não se lembra do *cookie* ou de qualquer outra informação de estado correspondente ao SYN.
- Se o cliente for verdadeiro, então um segmento ACK retornará. O servidor, ao receber esse ACK, precisa verificar se ele corresponde a algum SYN enviado antes. Como isto é feito se ele não guarda nenhuma memória sobre os segmentos SYN? Como você deve ter imaginado, o processo é realizado com o *cookie*. Para um ACK legítimo, em especial, o valor no campo de reconhecimento é igual ao número de sequência no SYNACK (o valor do *cookie*, neste caso) mais um (veja Figura 3.39). O servidor, então, executará a mesma função utilizando os mesmos campos no segmento ACK (que são os mesmos que no SYN original) e número secreto. Se o resultado da função mais um for o mesmo que o número de reconhecimento (*cookie*) no SYNACK do cliente, o servidor conclui que o ACK corresponde a um segmento SYN anterior e, portanto, é válido. O servidor, então, cria uma conexão totalmente aberta com um *socket*.
- Por outro lado, se o cliente não retorna um segmento ACK, então o SYN original não causou nenhum dano ao servidor, uma vez que este não alocou nenhum recurso em resposta ao SYN falso original.

Nossa discussão anterior concluiu que o cliente e o servidor estão preparados para se comunicar, isto é, que o servidor está ouvindo na porta pela qual o cliente envia seu segmento SYN. Vamos considerar o que acontece quando um hospedeiro recebe um segmento TCP cujos números de porta ou endereço IP não são compatíveis com nenhum dos *sockets* existentes no hospedeiro. Por exemplo, imagine que um hospedeiro receba um pacote TCP SYN com porta de destino 80, mas não está aceitando conexões nessa porta (isto é, não está rodando um servidor Web na porta 80). Então, ele enviará à origem um segmento especial de reinicialização. Esse segmento TCP tem o bit de *flag* RST ajustado para 1 (veja Seção 3.5.2). Assim, quando um hospedeiro envia um segmento de reinicialização ele está dizendo à origem: "Eu não tenho um *socket* para esse segmento. Favor não enviá-lo novamente". Quando um hospedeiro recebe um pacote UDP cujo número de porta de destino não é compatível com as portas de um UDP em curso, ele envia um datagrama ICMP especial, como será discutido no Capítulo 4.

Agora que obtivemos uma boa compreensão sobre gerenciamento da conexão TCP, vamos voltar à ferramenta de varredura de porta nmap e analisar mais precisamente como ela funciona. Para explorar uma porta TCP,

digamos que a porta 6789, o nmap enviará ao computador-alvo um segmento TCP SYN com a porta de destino. Os três possíveis resultados são:

- *O computador de origem recebe um segmento TCP SYNACK de um computador-alvo.* Como isso significa que uma aplicação está sendo executada com a porta TCP 6789 no computador-alvo, o nmap retorna "aberto".
- *O computador de origem recebe um segmento TCP RST de um computador-alvo.* Isto significa que o segmento SYN atingiu o computador-alvo, mas este não está executando uma aplicação com a porta TCP 6789. Mas o atacante, pelo menos, sabe que os segmentos destinados ao computador na porta 6789 não estão bloqueados pelo *firewall* no percurso entre o computador de origem e o alvo. (*Firewalls* são abordados no Capítulo 8.)
- *A origem não recebe nada.* Isto, provavelmente, significa que o segmento SYN foi bloqueado pelo firewall e nunca atingiu o computador-alvo.

O nmap é uma ferramenta potente, que pode "examinar o local" não só para abrir portas TCP, mas também para abrir portas UDP, para *firewalls* e suas configurações, e até mesmo para as versões de aplicações e sistemas operacionais. A maior parte é feita pela manipulação dos segmentos de gerenciamento da conexão TCP [Skoudis, 2006]. É possível fazer *download* do nmap pelo site <www.nmap.org>.

Com isso, concluímos nossa introdução ao controle de erro e controle de fluxo em TCP. Voltaremos ao TCP na Seção 3.7 e então examinaremos em mais detalhes o controle de congestionamento do TCP. Antes, contudo, vamos analisar a questão do controle de congestionamento em um contexto mais amplo.

3.6 PRINCÍPIOS DE CONTROLE DE CONGESTIONAMENTO

Nas seções anteriores, examinamos os princípios gerais e também os mecanismos específicos do TCP usados para prover um serviço de transferência confiável de dados em relação à perda de pacotes. Mencionamos antes que, na prática, essa perda resulta, de modo característico, de uma saturação de buffers de roteadores à medida que a rede fica congestionada. Assim, a retransmissão de pacotes trata de um sintoma de congestionamento de rede (a perda de um segmento específico de camada de transporte), mas não trata da causa do congestionamento da rede: demasiadas fontes tentando enviar dados a uma taxa muito alta. Para tratar da causa do congestionamento de rede, são necessários mecanismos para regular os remetentes quando ele ocorre.

Nesta seção, consideramos o problema do controle de congestionamento em um contexto geral, buscando entender por que ele é algo ruim, como o congestionamento de rede se manifesta no desempenho recebido por aplicações da camada superior e várias medidas que podem ser adotadas para evitá-lo ou reagir a ele. Esse estudo mais geral do controle de congestionamento é apropriado, já que, como acontece com a transferência confiável de dados, o congestionamento é um dos "dez mais" da lista de problemas fundamentalmente importantes no trabalho em rede. Concluímos esta seção com uma discussão sobre o controle de congestionamento no serviço de **taxa de bits disponível** (*available bit-rate* — ABR) em redes com **modo de transferência assíncrono** (ATM). A seção seguinte contém um estudo detalhado do algoritmo de controle de congestionamento do TCP.

3.6.1 As causas e os custos do congestionamento

Vamos começar nosso estudo geral do controle de congestionamento examinando três cenários de complexidade crescente nos quais ele ocorre. Em cada caso, examinaremos, primeiro, por que ele ocorre e, depois, seu custo (no que se refere aos recursos não utilizados integralmente e ao baixo desempenho recebido pelos sistemas finais). Não focalizaremos (ainda) como reagir a ele, ou evitá-lo; preferimos estudar uma questão mais simples, que é entender o que acontece quando hospedeiros aumentam sua taxa de transmissão e a rede fica congestionada.

Cenário 1: dois remetentes, um roteador com buffers infinitos

Começamos considerando o que talvez seja o cenário de congestionamento mais simples possível: dois hospedeiros (A e B), cada um conforme uma conexão que compartilha um único trecho de rede entre a origem e o destino, conforme mostra a Figura 3.43.

Vamos admitir que a aplicação no hospedeiro A esteja enviando dados para a conexão (por exemplo, passando dados para o protocolo de camada de transporte por um *socket*) a uma taxa média de λ_{in} bytes/s. Esses dados são originais no sentido de que cada unidade de dados é enviada para dentro do *socket* apenas uma vez. O protocolo de camada de transporte subjacente é simples. Os dados são encapsulados e enviados; não há recuperação de erros (por exemplo, retransmissão), controle de fluxo, nem controle de congestionamento. Desprezando a sobrecarga adicional causada pela adição de informações de cabeçalhos de camada de transporte e de camadas mais baixas, a taxa à qual o hospedeiro A oferece tráfego ao roteador nesse primeiro cenário é λ_{in} bytes/s. O hospedeiro B funciona de maneira semelhante, e admitimos, por simplicidade, que ele também esteja enviando dados a uma taxa de λ_{in} bytes/s. Os pacotes dos hospedeiros A e B passam por um roteador e por um enlace de saída compartilhado de capacidade R. O roteador tem buffers que lhe permitem armazenar os pacotes que chegam quando a taxa de chegada excede a capacidade do enlace de saída. No primeiro cenário, admitimos que o roteador tenha capacidade de armazenamento infinita.

A Figura 3.44 apresenta o desempenho da conexão do hospedeiro A nesse primeiro cenário. O gráfico da esquerda mostra a **vazão por conexão** (número de bytes por segundo no destinatário) como uma função da taxa de envio da conexão. Para uma taxa de transmissão entre 0 e $R/2$, a vazão no destinatário é igual à velocidade de envio do remetente — tudo o que este envia é recebido no destinatário com um atraso finito. Quando a velocidade de envio estiver acima de $R/2$, contudo, a vazão será somente $R/2$. Esse limite superior da vazão é consequência do compartilhamento da capacidade do enlace entre duas conexões. O enlace simplesmente não consegue entregar os pacotes a um destinatário com uma taxa em estado constante que exceda $R/2$. Não importa quão altas sejam as taxas de envio ajustadas nos hospedeiros A e B, eles jamais alcançarão uma vazão maior do que $R/2$.

Alcançar uma vazão de $R/2$ por conexão pode até parecer uma coisa boa, pois o enlace está sendo integralmente utilizado para entregar pacotes no destinatário. No entanto, o gráfico do lado direito da Figura 3.44 mostra as consequências de operar próximo à capacidade máxima do enlace. À medida que a taxa de envio se aproxima de $R/2$ (partindo da esquerda), o atraso médio fica cada vez maior. Quando a taxa de envio ultrapassa $R/2$, o número médio de pacotes na fila no roteador é ilimitado e o atraso médio entre a fonte e o destino se torna infinito (admitindo que as conexões operem a essas velocidades de transmissão durante um período de tempo infinito e que a capacidade de armazenamento também seja infinita). Assim, embora operar a uma vazão agregada próxima a R possa ser ideal do ponto de vista da vazão, está bem longe de ser ideal do ponto de vista do atraso. *Mesmo nesse cenário (extremamente) idealizado, já descobrimos um custo da rede congestionada — há grandes atrasos de fila quando a taxa de chegada de pacotes se aproxima da capacidade do enlace.*

FIGURA 3.43 CENÁRIO DE CONGESTIONAMENTO 1: DUAS CONEXÕES COMPARTILHANDO UM ÚNICO ROTEADOR COM NÚMERO INFINITO DE BUFFERS

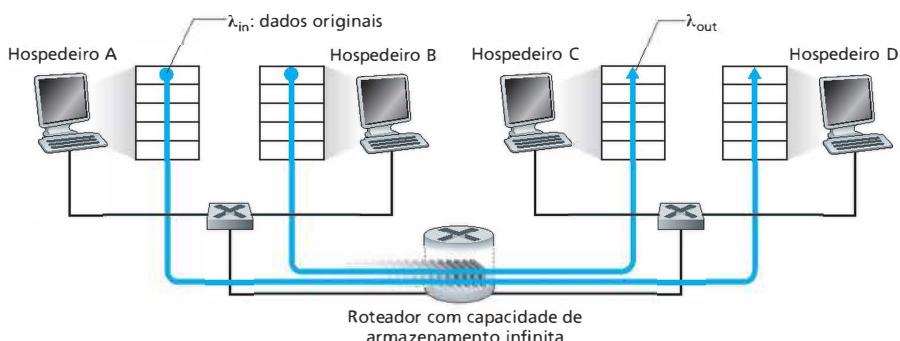
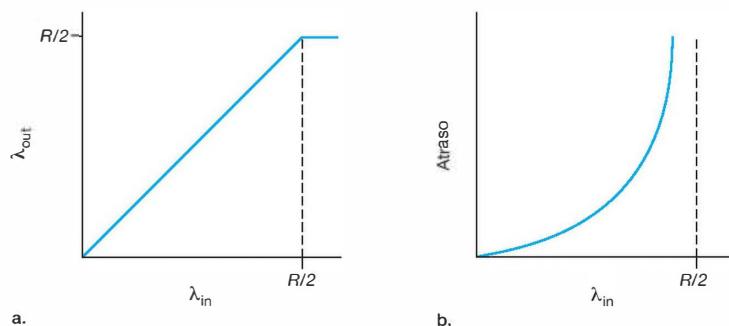



FIGURA 3.44 CENÁRIO DE CONGESTIONAMENTO 1: VAZÃO E ATRASO EM FUNÇÃO DA TAXA DE ENVIO DO HOSPEDEIRO

Cenário 2: dois remetentes, um roteador com buffers finitos

Vamos agora modificar um pouco o cenário 1 dos dois modos seguintes (veja a Figura 3.45). Primeiro, admitamos que a capacidade de armazenamento do roteador seja finita. Em uma situação real, essa suposição teria como consequência o descarte de pacotes que chegam a um buffer que já está cheio. Segundo, admitamos que cada conexão seja confiável. Se um pacote contendo um segmento de camada de transporte for descartado no roteador, o remetente por fim o retransmitirá. Como os pacotes podem ser retransmitidos, agora temos de ser mais cuidadosos com o uso da expressão "taxa de envio". Especificamente, vamos de novo designar a taxa com que a aplicação envia dados originais para dentro do *socket* como λ_{in} bytes/s. A taxa com que a camada de transporte envia segmentos (contendo dados originais *e* dados retransmitidos) para dentro da rede será denominada λ'_{in} bytes/s. Essa taxa (λ'_{in}) às vezes é denominada **carga oferecida** à rede.

O desempenho obtido no cenário 2 agora dependerá muito de como a retransmissão é realizada. Primeiro, considere o caso não realista em que o hospedeiro A consiga, de algum modo (fazendo mágica!), determinar se um buffer do roteador está livre no roteador e, portanto, envia um pacote apenas quando um buffer estiver livre. Nesse caso, não ocorreria nenhuma perda, λ_{in} seria igual a λ'_{in} e a vazão da conexão seria igual a λ_{in}. Esse caso é mostrado pela curva superior da Figura 3.46(a). Do ponto de vista da vazão, o desempenho é ideal — tudo o que

FIGURA 3.45 CENÁRIO 2: DOIS HOSPEDEIROS (COM RETRANSMISSÕES) E UM ROTEADOR COM BUFFERS FINITOS

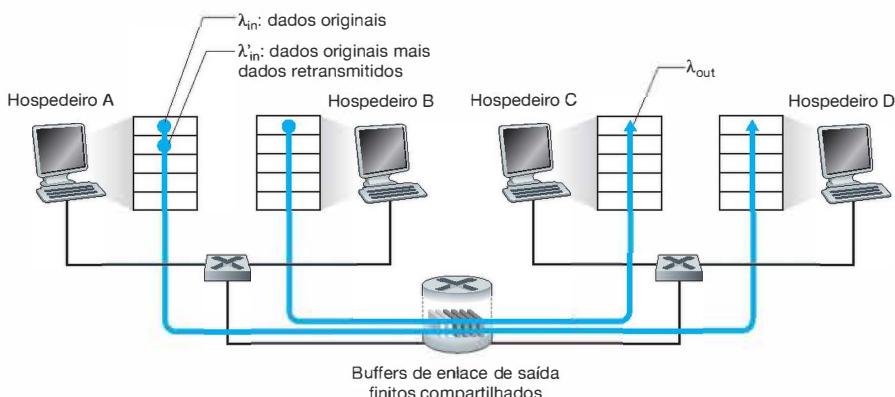
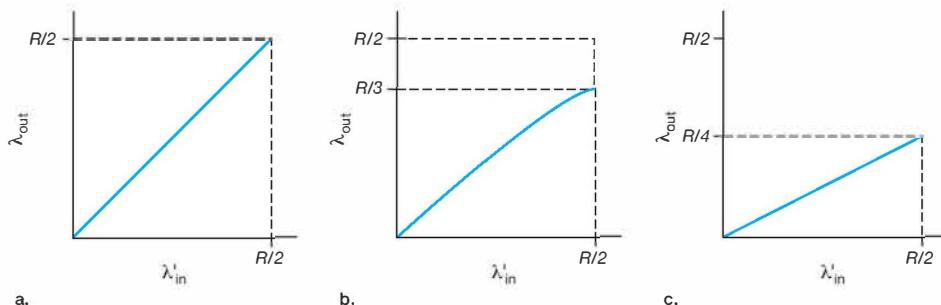
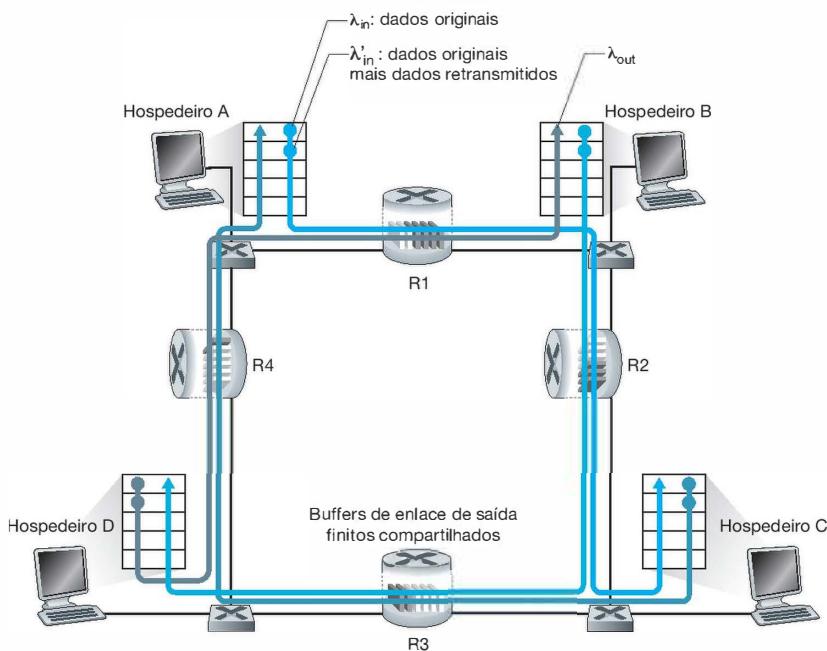



FIGURA 3.46 DESEMPENHO NO CENÁRIO 2 COM BUFFERS FINITOS

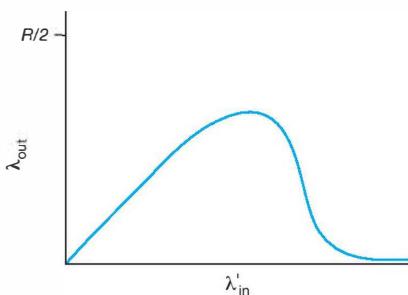
é enviado é recebido. Note que, nesse cenário, a taxa média de envio do hospedeiro não pode ultrapassar $R/2$, já que admitimos que nunca ocorre perda de pacote.

Considere, em seguida, o caso um pouco mais realista em que o remetente retransmite apenas quando sabe, com certeza, que o pacote foi perdido. (De novo, essa suposição é um pouco forçada. Contudo, é possível ao hospedeiro remetente ajustar seu temporizador de retransmissão para uma duração longa o suficiente para ter razoável certeza de que um pacote que não foi reconhecido foi perdido.) Nesse caso, o desempenho pode ser parecido com o mostrado na Figura 3.46(b). Para avaliar o que está acontecendo aqui, considere o caso em que a carga oferecida, λ'_{in} (a taxa de transmissão dos dados originais mais as retransmissões), é igual a $R/2$. De acordo com a Figura 3.46(b), nesse valor de carga oferecida, a velocidade com a qual os dados são entregues à aplicação do destinatário é $R/3$. Assim, de $0{,}5R$ unidade de dados transmitida, $0{,}333R$ byte/s (em média) são dados originais e $0{,}166R$ byte/s (em média) são dados retransmitidos. *Observamos aqui outro custo de uma rede congestionada — o remetente deve realizar retransmissões para compensar os pacotes descartados (perdidos) pelo estouro do buffer.*

Finalmente, vamos considerar o caso em que a temporização do remetente se esgote prematuramente e ele retransmita um pacote que ficou atrasado na fila, mas que ainda não está perdido. Aqui, tanto o pacote de dados original quanto a retransmissão podem alcançar o destinatário. É claro que o destinatário precisa apenas de uma cópia desse pacote e descartará a retransmissão. Nesse caso, o trabalho realizado pelo roteador ao repassar a cópia retransmitida do pacote original é desperdiçado, pois o destinatário já terá recebido a cópia original do pacote. Em vez disso, seria melhor o roteador usar a capacidade de transmissão do enlace para enviar um pacote diferente. *Eis aqui mais um custo da rede congestionada — retransmissões desnecessárias feitas pelo remetente em face de grandes atrasos podem fazer que um roteador use sua largura de banda de enlace para repassar cópias desnecessárias de um pacote.* A Figura 3.4.6(c) mostra a vazão *versus* a carga oferecida admitindo-se que cada pacote seja enviado (em média) duas vezes pelo roteador. Visto que cada pacote é enviado duas vezes, a vazão terá um valor assintótico de $R/4$ à medida que a carga oferecida se aproximar de $R/2$.

Cenário 3: quatro remetentes, roteadores com buffers finitos e trajetos com múltiplos roteadores

Em nosso cenário final de congestionamento, quatro hospedeiros transmitem pacotes sobre trajetos sobrepostos que apresentam dois saltos, como ilustrado na Figura 3.47. Novamente admitamos que cada hospedeiro use um mecanismo de temporização/retransmissão para executar um serviço de transferência confiável de dados, que todos os hospedeiros tenham o mesmo valor de λ_{in} e que todos os enlaces dos roteadores tenham capacidade de R bytes/s.

Vamos considerar a conexão do hospedeiro A ao hospedeiro C que passa pelos roteadores R1 e R2. A conexão A–C compartilha o roteador R1 com a conexão D–B e o roteador 2 com a conexão B–D. Para valores extremamente pequenos de λ_{in}, esgotamentos de buffers são raros (como acontecia nos cenários de congestionamento 1 e 2) e

FIGURA 3.47 QUATRO REMETENTES, ROTEADORES COM BUFFERS FINITOS E TRAJETOS COM VÁRIOS SALTOS

a vazão é quase igual à carga oferecida. Para valores de λ_{in} um pouco maiores, a vazão correspondente é também maior, pois mais dados originais estão sendo transmitidos para a rede e entregues no destino, e os esgotamentos ainda são raros. Assim, para valores pequenos de λ_{in}, um aumento em λ_{in} resulta em um aumento em λ_{out}.

Como já analisamos o caso de tráfego extremamente baixo, vamos examinar aquele em que λ_{in} (e, portanto, λ'_{in}) é extremamente alto. Considere o roteador R2. O tráfego A–C que chega ao roteador R2 (após ter sido repassado de R1) pode ter uma taxa de chegada em R2 de, no máximo, *R*, que é a capacidade do enlace de R1 a R2, não importando qual seja o valor de λ_{in}. Se λ'_{in} for extremamente alto para todas as conexões (incluindo a conexão B–D), então a taxa de chegada do tráfego B–D em R2 poderá ser muito maior do que a taxa do tráfego A–C. Como os tráfegos A–C e B–D têm de competir no roteador R2 pelo espaço limitado de buffer, a quantidade de tráfego A–C que consegue passar por R2 (isto é, que não se perde pelo congestionamento de buffer) diminui cada vez mais à medida que a carga oferecida de B–D vai ficando maior. No limite, quando a carga oferecida se aproxima do infinito, um buffer vazio em R2 é logo preenchido por um pacote B–D e a vazão da conexão A–C em R2 cai a zero. Isso, por sua vez, *implica que a vazão fim a fim de A–C vai a zero* no limite de tráfego pesado. Essas considerações dão origem ao comportamento da carga oferecida *versus* a vazão mostrada na Figura 3.48.

A razão para o decréscimo final da vazão com o crescimento da carga oferecida é evidente quando consideramos a quantidade de trabalho desperdiçado realizado pela rede. No cenário de alto tráfego que acabamos de descrever, sempre que um pacote é descartado em um segundo roteador, o trabalho realizado pelo primeiro para enviar o pacote ao segundo acaba sendo "desperdiçado". A rede teria se saído igualmente bem (melhor dizendo, igualmente mal) se o primeiro roteador tivesse apenas descartado aquele pacote e ficado inativo. Sendo mais objetivos, a capacidade de transmissão utilizada no primeiro roteador para enviar o pacote ao segundo teria sido maximizada para transmitir um pacote diferente. (Por exemplo, ao selecionar um pacote para transmissão, seria melhor que um roteador desse prioridade a pacotes que já atravessaram alguns roteadores anteriores.) *Portanto,*

vemos aqui mais um custo, o do descarte de pacotes devido ao congestionamento — quando um pacote é descartado ao longo de um caminho, a capacidade de transmissão que foi usada em cada um dos enlaces anteriores para repassar o pacote até o ponto em que foi descartado acaba sendo desperdiçada.

FIGURA 3.48 DESEMPENHO OBTIDO NO CENÁRIO 3, COM BUFFERS FINITOS E TRAJETOS COM MÚLTIPLOS ROTEADORES

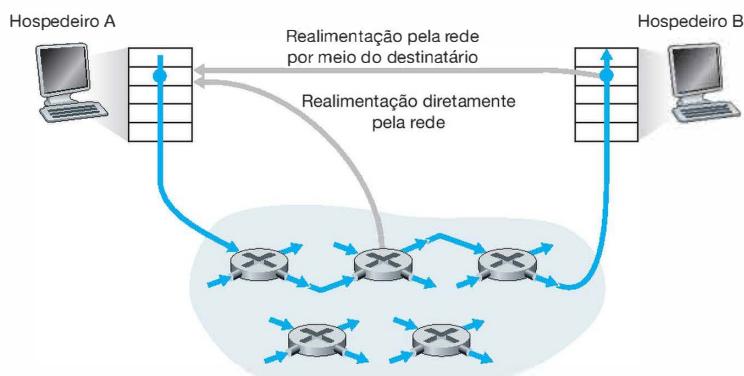
3.6.2 Mecanismos de controle de congestionamento

Na Seção 3.7, examinaremos em detalhes os mecanismos específicos do TCP para o controle de congestionamento. Aqui, identificaremos os dois procedimentos mais comuns adotados, na prática, para esse controle. Além disso, examinaremos arquiteturas específicas de rede e protocolos de controle que incorporam tais procedimentos.

No nível mais amplo, podemos distinguir mecanismos de controle de congestionamento conforme a camada de rede ofereça ou não assistência explícita à camada de transporte com a finalidade de controle de congestionamento:

- *Controle de congestionamento fim a fim.* Nesse método, a camada de rede não fornece *nenhum suporte explícito* à camada de transporte com a finalidade de controle de congestionamento. Até mesmo a presença de congestionamento na rede deve ser intuída pelos sistemas finais com base apenas na observação do comportamento da rede (por exemplo, perda de pacotes e atraso). Veremos na Seção 3.7 que o TCP deve necessariamente adotar esse método fim a fim para o controle de congestionamento, uma vez que a camada IP não fornece realimentação de informações aos sistemas finais quanto ao congestionamento da rede. A perda de segmentos TCP (apontada por uma temporização ou por três reconhecimentos duplicados) é tomada como indicação de congestionamento de rede, e o TCP reduz o tamanho da janela de acordo com isso. Veremos também que as novas propostas para o TCP usam valores de atraso de viagem de ida e volta crescentes como indicadores de aumento do congestionamento da rede.
- *Controle de congestionamento assistido pela rede.* Com esse método, os componentes da camada de rede (isto é, roteadores) fornecem retroalimentação específica de informações ao remetente a respeito do estado de congestionamento na rede. Essa retroalimentação pode ser tão simples como um único bit indicando o congestionamento em um enlace. Adotada nas primeiras arquiteturas de rede IBM SNA [Schwartz, 1982] e DEC DECnet [Jain, 1989; Ramakrishnan, 1990], essa abordagem foi proposta recentemente para redes TCP/IP [Floyd TCP, 1994; RFC 3168]; ela é usada também no controle de congestionamento em ATM com serviço de transmissão ABR (Available Bit Rate), como discutido a seguir. A retroalimentação mais sofisticada de rede também é possível. Por exemplo, uma forma de controle de congestionamento ATM ABR que estudaremos mais adiante permite que um roteador informe explicitamente ao remetente a velocidade de transmissão que ele (o roteador) pode suportar em um enlace de saída. O protocolo XCP [Katabi, 2002] provê um retorno (*feedback*) calculado pelo

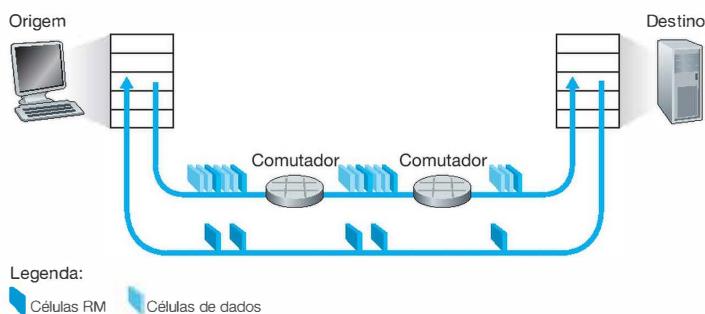
roteador para cada origem, transmitido no cabeçalho do pacote, referente ao modo que essa origem deve aumentar ou diminuir sua taxa de transmissão.

Para controle de congestionamento assistido pela rede, a informação sobre congestionamento é em geral realimentada da rede para o remetente por um de dois modos, como ilustra a Figura 3.49. Retroalimentação direta pode ser enviada de um roteador de rede ao remetente. Esse modo de notificação em geral toma a forma de um **pacote de congestionamento** (*choke packet*) (que, em essência, diz: "Estou congestionado!"). O segundo modo de notificação ocorre quando um roteador marca/atualiza um campo em um pacote que está fluindo do remetente ao destinatário para indicar congestionamento. Ao receber um pacote marcado, o destinatário informa ao remetente a indicação de congestionamento. Note que esse último modo de notificação leva, no mínimo, o tempo total de uma viagem de ida e volta.

3.6.3 Exemplo de controle de congestionamento assistido pela rede: controle de congestionamento ATM ABR

Concluímos esta seção com um breve estudo de caso do algoritmo de controle de congestionamento ATM ABR — um protocolo que usa uma técnica de controle assistido pela rede. Salientamos que nossa meta aqui não é, de modo algum, descrever em detalhes aspectos da arquitetura ATM, mas mostrar um protocolo que adota uma abordagem de controle de congestionamento notavelmente diferente da adotada pelo protocolo TCP da Internet. Na verdade, apresentamos a seguir apenas os poucos aspectos da arquitetura ATM necessários para entender o controle de congestionamento ABR.

Na essência, o ATM adota uma abordagem orientada para circuito virtual (CV) da comutação de pacotes. Lembre, da nossa discussão no Capítulo 1, que isso significa que cada comutador no caminho entre a origem e o destino manterá estado sobre o CV entre a origem e o destino. Esse estado por CV permite que um comutador monitore o comportamento de remetentes individuais (por exemplo, monitorando sua taxa média de transmissão) e realize ações específicas de controle (tais como sinalizar de modo explícito ao remetente para que ele reduza sua taxa quando o comutador fica congestionado). Esse estado por CV em comutadores de rede torna o ATM idealmente adequado para realizar controle de congestionamento assistido pela rede.

O ABR foi projetado como um serviço de transferência de dados elástico que faz lembrar, de certo modo, o TCP. Quando a rede está vazia, o serviço ABR deve ser capaz de aproveitar a vantagem da largura de banda disponível. Quando está congestionada, deve limitar sua taxa de transmissão a algum valor mínimo predeterminado. Um tutorial detalhado sobre controle de congestionamento e gerenciamento de tráfego ATM ABR é fornecido por [Jain, 1996].

FIGURA 3.49 DOIS CAMINHOS DE REALIMENTAÇÃO PARA INFORMAÇÃO SOBRE CONGESTIONAMENTO INDICADO PELA REDE

A Figura 3.50 mostra a estrutura do controle de congestionamento para ATM ABR. Nessa discussão, adotaremos a terminologia ATM (por exemplo, usaremos o termo "comutador" em vez de "roteador" e o termo "célula" em vez de "pacote"). Com o serviço ATM ABR, as células de dados são transmitidas de uma origem a um destino por meio de uma série de comutadores intermediários. Intercaladas às células de dados estão as **células de gerenciamento de recursos**, ou **células RM** (*resource-management*); elas podem ser usadas para portar informações relacionadas ao congestionamento entre hospedeiros e comutadores. Quando uma célula RM chega a um destinatário, ela será "virada" e enviada de volta ao remetente (possivelmente após o destinatário ter modificado o conteúdo dela). Também é possível que um comutador gere, ele mesmo, uma célula RM e a envie direto a uma origem. Assim, células RM podem ser usadas para fornecer realimentação diretamente da rede e também realimentação da rede por intermédio do destinatário, como mostra a Figura 3.50.

O controle de congestionamento ATM ABR é um método baseado em taxa. O remetente estima explicitamente uma taxa máxima na qual pode enviar e se autorregula de acordo com ela. O serviço ABR provê três mecanismos para sinalizar informações relacionadas a congestionamento dos comutadores ao destinatário:

- *Bit EFCI*. Cada *célula de dados* contém um **bit EFCI de indicação explícita de congestionamento** à frente (*explicit forward congestion indication*). Um comutador de rede congestionado pode modificar o bit EFCI dentro de uma célula para 1, a fim de sinalizar congestionamento ao hospedeiro destinatário, que deve verificar o bit EFCI em todas as células de dados recebidas. Quando uma célula RM chega ao destinatário, se a célula de dados recebida mais recentemente tiver o bit EFCI com valor 1, o destinatário modifica o bit de indicação de congestionamento (o bit CI) da célula RM para 1 e envia a célula RM de volta ao remetente. Usando o bit EFCI em células de dados e o bit CI em células RM, um remetente pode ser notificado sobre congestionamento em um comutador da rede.
- *Bits CI e NI*. Como já foi observado, células RM remetente/destinatário estão intercaladas com células de dados. A taxa de intercalação de células RM é um parâmetro ajustável, sendo uma célula RM a cada 32 células de dados o valor *default*. Essas células RM têm um bit de indicação de congestionamento (**CI**) e um bit **NI** (*no increase* — não aumentar) que podem ser ajustados por um comutador de rede congestionado. Especificamente, um comutador pode modificar para 1 o bit NI de uma célula RM que está passando quando a condição de congestionamento é leve e pode modificar o bit CI para 1 em severas condições de congestionamento. Quando um hospedeiro destinatário recebe uma célula RM, ele a envia de volta ao remetente com seus bits CI e NI inalterados (exceto que o CI pode ser ajustado para 1 pelo destinatário como resultado do mecanismo EFCI descrito antes).
- *Ajuste de ER*. Cada célula RM contém também um **campo ER** (*explicit rate* — taxa explícita) de dois bytes. Um comutador congestionado pode reduzir o valor contido no campo ER de uma célula RM que está passando. Desse modo, o campo ER será ajustado para a taxa mínima suportável por todos os comutadores no trajeto origem-destino.

FIGURA 3.50 ESTRUTURA DE CONTROLE DE CONGESTIONAMENTO PARA O SERVIÇO ATM ABR

Uma origem ATM ABR ajusta a taxa na qual pode enviar células como uma função dos valores de CI, NI e ER de uma célula RM devolvida. As regras para fazer esse ajuste são bastante complicadas e um tanto tediosas. O leitor poderá consultar Jain [1996] se quiser saber mais detalhes.

3.7 CONTROLE DE CONGESTIONAMENTO NO TCP

Nesta seção, voltamos ao estudo do TCP. Como aprendemos na Seção 3.5, o TCP provê um serviço de transferência confiável entre dois processos que rodam em hospedeiros diferentes. Outro componente de extrema importância do TCP é seu mecanismo de controle de congestionamento. Como indicamos na seção anterior, o TCP deve usar controle de congestionamento fim a fim em vez de controle assistido pela rede, já que a camada IP não fornece aos sistemas finais realimentação explícita relativa ao congestionamento da rede.

A abordagem adotada pelo TCP é obrigar cada remetente a limitar a taxa à qual enviam tráfego para sua conexão como uma função do congestionamento de rede percebido. Se um remetente TCP perceber que há pouco congestionamento no caminho entre ele e o destinatário, aumentará sua taxa de envio; se perceber que há congestionamento, reduzirá sua taxa de envio. Mas essa abordagem levanta três questões. Primeiro, como um remetente TCP limita a taxa pela qual envia tráfego para sua conexão? Segundo, como um remetente TCP percebe que há congestionamento entre ele e o destinatário? E terceiro, que algoritmo o remetente deve utilizar para modificar sua taxa de envio como uma função do congestionamento fim a fim percebido?

Para começar, vamos examinar como um remetente TCP limita a taxa de envio à qual envia tráfego para sua conexão. Na Seção 3.5, vimos que cada lado de uma conexão TCP consiste em um buffer de recepção, um buffer de envio e diversas variáveis (`LastByteRead`, `rwnd` e assim por diante). O mecanismo de controle de congestionamento que opera no remetente monitora uma variável adicional, a **janela de congestionamento**. Esta, denominada `cwnd`, impõe uma limitação à taxa à qual um remetente TCP pode enviar tráfego para dentro da rede. Especificamente, a quantidade de dados não reconhecidos em um hospedeiro não pode exceder o mínimo de `cwnd` e `rwnd`, ou seja:

```
LastByteSent – LastByteAcked ≤ min{cwnd, rwnd}
```

Para concentrar a discussão no controle de congestionamento (ao contrário do controle de fluxo), daqui em diante vamos admitir que o buffer de recepção TCP seja tão grande que a limitação da janela de recepção pode ser desprezada; assim, a quantidade de dados não reconhecidos no remetente estará limitada apenas por `cwnd`. Vamos admitir também que o remetente sempre tenha dados para enviar, isto é, que todos os segmentos dentro da janela de congestionamento sejam enviados.

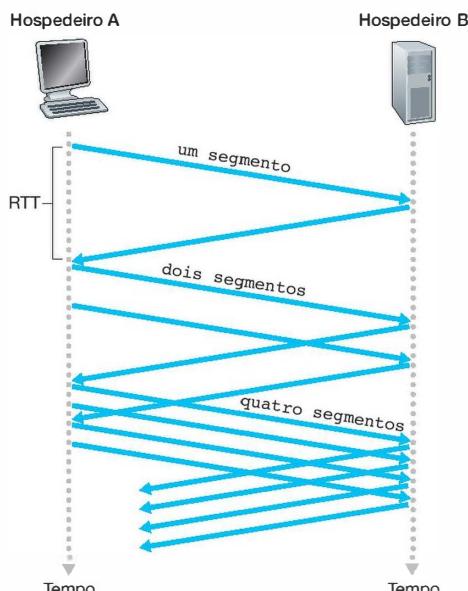
A restrição citada limita a quantidade de dados não reconhecidos no remetente e, por conseguinte, limita indiretamente a taxa de envio do remetente. Para entender melhor, considere uma conexão na qual perdas e atrasos de transmissão de pacotes sejam desprezíveis. Então, em linhas gerais, no início de cada RTT a limitação permite que o remetente envie `cwnd` bytes de dados para a conexão; ao final do RTT, o remetente recebe reconhecimentos para os dados. *Assim, a taxa de envio do remetente é aproximadamente* `cwnd/RTT` *bytes por segundo. Portanto, ajustando o valor de* `cwnd`*, o remetente pode ajustar a taxa à qual envia dados para sua conexão.*

Em seguida, vamos considerar como um remetente TCP percebe que há congestionamento no caminho entre ele e o destino. Definimos "evento de perda" em um remetente TCP como a ocorrência de um esgotamento de temporização (*timeout*) ou do recebimento de três ACKs duplicados do destinatário (lembre-se da nossa discussão, na Seção 3.5.4, do evento de temporização apresentado na Figura 3.33 e da subsequente modificação para incluir transmissão rápida quando do recebimento de três ACKs duplicados). Quando há congestionamento excessivo, então um (ou mais) buffers de roteadores ao longo do caminho transborda, fazendo que um datagrama (contendo um segmento TCP) seja descartado. O datagrama descartado, por sua vez, resulta em um evento de perda no remetente — ou um esgotamento de temporização ou o recebimento de três ACKs duplicados — que é tomado por ele como uma indicação de congestionamento no caminho remetente-destinatário.

Já consideramos como é detectado o congestionamento; agora vamos analisar o caso mais otimista de uma rede livre de congestionamento, isto é, quando não ocorre um evento de perda. Nesse caso, o TCP remetente receberá reconhecimentos para segmentos não reconhecidos antes. Como veremos, o TCP considerará a chegada desses reconhecimentos como uma indicação de que tudo está bem — os segmentos que estão sendo transmitidos para a rede estão sendo entregues com sucesso no destinatário — e usará reconhecimentos para aumentar o tamanho de sua janela de congestionamento (e, por conseguinte, sua taxa de transmissão). Note que, se os reconhecimentos chegarem ao remetente a uma taxa relativamente baixa (por exemplo, se o atraso no caminho fim a fim for alto ou se nele houver um enlace de baixa largura de banda), então a janela de congestionamento será aumentada a uma taxa um tanto baixa. Por outro lado, se os reconhecimentos chegarem a uma taxa alta, então a janela de congestionamento será aumentada mais depressa. Como o TCP utiliza reconhecimentos para acionar (ou regular) o aumento de tamanho de sua janela de congestionamento, diz-se que o TCP é **autorregulado**.

Dado o *mecanismo* de ajustar o valor de cwnd para controlar a taxa de envio, permanece a importante pergunta: *Como* um remetente TCP deve determinar a taxa a que deve enviar? Se os remetentes TCP enviam coletivamente muito rápido, eles podem congestionar a rede, levando ao tipo de congestionamento que vimos na Figura 3.48. De fato, a versão de TCP que vamos estudar de modo breve foi desenvolvida em resposta aos congestionamentos da Internet observados em versões anteriores do TCP [Jacobson, 1988]. Entretanto, se os remetentes forem muito cautelosos e enviarem lentamente, eles podem subutilizar a largura de banda na rede; ou seja, os remetentes TCP podem enviar a uma taxa mais alta sem congestionar a rede. Então, como os remetentes TCP determinam suas taxas de envio de um modo que não congestionem mas, ao mesmo tempo, façam uso de toda a largura de banda? Os remetentes TCP são claramente coordenados, ou existe uma abordagem distribuída na qual eles podem ajustar suas taxas de envio baseando-se apenas nas informações locais? O TCP responde a essas perguntas utilizando os seguintes princípios:

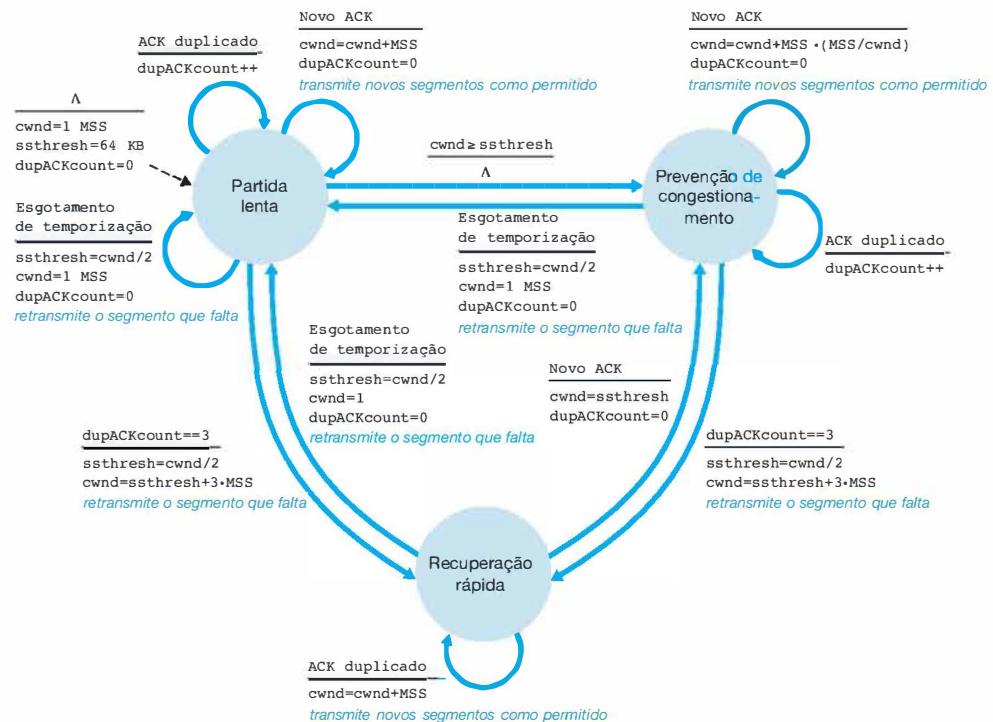
- *Um segmento perdido implica congestionamento, portanto, a taxa do remetente TCP deve diminuir quando um segmento é perdido.* Lembre-se da nossa discussão na Seção 3.5.4, de que um evento de esgotamento do temporizador ou o recebimento de quatro reconhecimentos para dado segmento (um ACK original e, depois, três ACKs duplicados) é interpretado como uma indicação de "evento de perda" absoluto do segmento subsequente ao ACK quadruplicado, acionando uma retransmissão do segmento perdido. De um ponto de vista do controle, a pergunta é como o remetente TCP deve diminuir sua janela de congestionamento e, portanto, sua taxa de envio, em resposta ao suposto evento de perda.
- *Um segmento reconhecido indica que a rede está enviando os segmentos do remetente ao destinatário e, por isso, a taxa do remetente pode aumentar quando um ACK chegar para um segmento não reconhecido antes.* A chegada de reconhecimentos é tida como uma indicação absoluta de que tudo está bem — os segmentos estão sendo enviados com sucesso do remetente ao destinatário, fazendo, assim, que a rede não fique congestionada. Dessa forma, o tamanho da janela de congestionamento pode ser elevado.
- *Busca por largura de banda.* Dados os ACKs que indicam um percurso de origem a destino sem congestionamento, e eventos de perda que indicam um percurso congestionado, a estratégia do TCP de ajustar sua taxa de transmissão é aumentar a taxa em resposta aos ACKs que chegam até que ocorra um evento de perda, momento em que a taxa de transmissão diminui. Desse modo, o remetente TCP aumenta sua taxa de transmissão para buscar a taxa pela qual o congestionamento se inicia, recua dela e de novo faz a busca para ver se a taxa de início do congestionamento foi alterada. O comportamento do remetente TCP é análogo a uma criança que pede (e ganha) cada vez mais doces até por fim receber um "Não!", recuar, mas começar a pedir novamente pouco tempo depois. Observe que não há nenhuma sinalização explícita de congestionamento pela rede — os ACKs e eventos de perda servem como sinais implícitos — e que cada remetente TCP atua em informações locais em momentos diferentes de outros.

Após essa visão geral sobre controle de congestionamento no TCP, agora podemos considerar os detalhes do renomado **algoritmo de controle de congestionamento TCP**, sendo primeiro descrito em Jacobson [1988] e padronizado em [RFC 5681]. O algoritmo possui três componentes principais: (1) partida lenta, (2) contenção

de congestionamento e (3) recuperação rápida. A partida lenta e a contenção de congestionamento são componentes obrigatórios do TCP, diferenciando em como eles aumentam o tamanho do cwnd em resposta a ACKs recebidos. Abordaremos em poucas palavras que a partida lenta aumenta o tamanho do cwnd de forma mais rápida (apesar do nome!) do que a contenção de congestionamento. A recuperação rápida é recomendada, mas não exigida, para remetentes TCP.

Partida lenta

Quando uma conexão TCP começa, o valor de cwnd costuma ser inicializado em 1 MSS [RFC 3390], resultando em uma taxa inicial de envio de aproximadamente MSS/RTT. Como exemplo, se MSS = 500 bytes e RTT = 200 ms, então a taxa de envio inicial resultante é cerca de 20 kbits/s apenas. Como a largura de banda disponível para a conexão pode ser muito maior do que MSS/RTT, o remetente TCP gostaria de aumentar a quantidade de largura de banda rapidamente. Dessa forma, no estado de partida lenta, o valor de cwnd começa em 1 MSS e aumenta 1 MSS toda vez que um segmento transmitido é reconhecido. No exemplo da Figura 3.51, o TCP envia o primeiro segmento para a rede e aguarda um reconhecimento. Quando este chega, o remetente TCP aumenta a janela de congestionamento em 1 MSS e envia dois segmentos de tamanho máximo. Esses segmentos são reconhecidos, e o remetente aumenta a janela de congestionamento em 1 MSS para cada reconhecimento de segmento, fornecendo uma janela de congestionamento de 4 MSS e assim por diante. Esse processo resulta em uma multiplicação da taxa de envio a cada RTT. Assim, a taxa de envio TCP se inicia lenta, mas cresce exponencialmente durante a fase de partida lenta.

Mas em que momento esse crescimento exponencial termina? A partida lenta apresenta diversas respostas para essa pergunta. Primeiro, se houver um evento de perda (ou seja, um congestionamento) indicado por um esgotamento de temporização, o remetente TCP estabelece o valor de cwnd em 1 e inicia o processo de partida lenta novamente. Ele também estabelece o valor de uma segunda variável de estado, ssthresh (abreviação de "*slow start threshold*" [limiar de partida lenta]), em cwnd/2 — metade do valor

FIGURA 3.51 PARTIDA LENTA TCP

da janela de congestionamento quando este foi detectado. O segundo modo pelo qual a partida lenta pode terminar é ligado diretamente ao valor de `ssthresh`. Visto que `ssthresh` é metade do valor de `cwnd` quando o congestionamento foi detectado pela última vez, pode ser uma atitude precipitada continuar duplicando `cwnd` ao atingir ou ultrapassar o valor de `ssthresh`. Assim, quando o valor de `cwnd` se igualar ao de `ssthresh`, a partida lenta termina e o TCP é alterado para o modo de prevenção de congestionamento. Como veremos, o TCP aumenta `cwnd` com mais cautela quando está no modo de prevenção de congestionamento. O último modo pelo qual a partida lenta pode terminar é se três ACKs duplicados forem detectados, caso no qual o TCP apresenta uma retransmissão rápida (veja Seção 3.5.4) e entra no estado de recuperação rápida, como discutido a seguir. O comportamento do TCP na partida lenta está resumido na descrição FSM do controle de congestionamento no TCP na Figura 3.52. O algoritmo de partida lenta foi de início proposto por Jacobson [1998]; uma abordagem semelhante à partida lenta também foi proposta de maneira independente em Jain [1986].

Prevenção de congestionamento

Ao entrar no estado de prevenção de congestionamento, o valor de `cwnd` é cerca de metade de seu valor quando o congestionamento foi encontrado pela última vez — o congestionamento poderia estar por perto! Desta forma, em vez de duplicar o valor de `cwnd` a cada RTT, o TCP adota um método mais conservador e aumenta o valor de `cwnd` por meio de um único MSS a cada RTT [RFC 5681]. Isso pode ser realizado de diversas formas. Uma abordagem comum é o remetente aumentar `cwnd` por MSS bytes (MSS/`cwnd`)no momento em que um novo reconhecimento chegar. Por exemplo, se o MSS possui 1.460 bytes e `cwnd`, 14.600 bytes, então

FIGURA 3.52 DESCRIÇÃO FSM DO CONTROLE DE CONGESTIONAMENTO NO TCP

10 segmentos estão sendo enviados dentro de um RTT. Cada ACK que chega (considerando um ACK por segmento) aumenta o tamanho da janela de congestionamento em 1/10 MSS e, assim, o valor da janela de congestionamento terá aumentado em 1 MSS após os ACKs quando todos os segmentos tiverem sido recebidos.

Mas em que momento o aumento linear da prevenção de congestionamento (de 1 MSS por RTT) deve terminar? O algoritmo de prevenção de congestionamento TCP se comporta da mesma forma quando ocorre um esgotamento de temporização. Como no caso da partida lenta: o valor de `cwnd` é ajustado para 1 MSS, e o valor de `ssthresh` é atualizado para metade do valor de `cwnd` quando ocorreu o evento de perda. Lembre-se, entretanto, de que um evento de perda também pode ser acionado por um evento ACK duplicado triplo. Neste caso, a rede continua a enviar segmentos do remetente ao destinatário (como indicado pelo recebimento de ACKs duplicados). Portanto, o comportamento do TCP para esse tipo de evento de perda deve ser menos drástico do que com uma perda de esgotamento de temporização: O TCP reduz o valor de `cwnd` para metade (adicionando em 3 MSS a mais para contabilizar os ACKs duplicados triplos recebidos) e registra o valor de `ssthresh` como metade do de `cwnd` quando os ACKs duplicados triplos foram recebidos. Então, entra-se no estado de recuperação rápida.

PRINCÍPIOS NA PRÁTICA

Divisão do TCP: otimizando o desempenho de serviços da nuvem

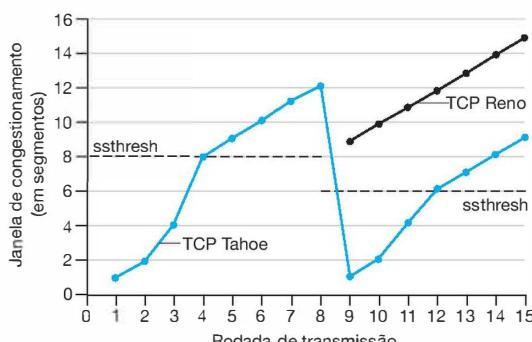
Para serviços da nuvem como busca, e-mail e redes sociais, deseja-se prover um alto nível de responsividade, de preferência dando aos usuários a ilusão de que os serviços estão rodando dentro de seus próprios sistemas finais (inclusive seus smartphones). Isso pode ser um grande desafio, pois os usuários em geral estão localizados distantes dos centros de dados que são responsáveis por servir o conteúdo dinâmico associado aos serviços da nuvem. Na verdade, se o sistema final estiver longe de um centro de dados, então o RTT será grande, potencialmente levando a um tempo de resposta maior, devido à partida lenta do TCP.

Como um estudo de caso, considere o atraso no recebimento de uma resposta para uma consulta. Em geral, o servidor requer três janelas TCP durante a partida lenta para entregar a resposta [Pathak, 2010]. Assim, o tempo desde que um sistema final inicia uma conexão TCP até o momento em que ele recebe o último pacote da resposta é cerca de 4 · RTT (um RTT para estabelecer a conexão TCP mais três RTTs para as três janelas de dados) mais o tempo de processamento no centro de dados. Esses atrasos de RTT podem levar a um atraso observável no retorno de resultados de busca para uma fração significativa de consultas. Além do mais, pode haver uma significativa perda de pacotes nas redes de acesso, ocasionando retransmissões do TCP e até mesmo atrasos maiores.

Um modo de aliviar esse problema e melhorar o desempenho percebido pelo usuário é (1) instalar servidores de *front-end* mais perto dos usuários e (2) utilizar a **divisão do TCP**, quebrando a conexão TCP no servidor de *front-end*. Com a divisão do TCP, o cliente estabelece uma conexão TCP com o *front-end* nas proximidades, e o *front-end* mantém uma conexão TCP persistente com o centro de dados, com uma janela de congestionamento TCP muito grande [Tariq, 2008; Pathak, 2010; Chen 2011]. Com essa técnica, o tempo de resposta torna-se cerca de $4 \cdot RTT_{FE} + RTT_{BE}$ + tempo de processamento, onde RTT_{FE} é o tempo de viagem de ida e volta entre cliente e servidor de *front-end*, e RTT_{BE} é o tempo de viagem de ida e volta entre o servidor de *front-end* e o centro de dados (servidor de *back-end*). Se o servidor de *front-end* estiver perto do cliente, o tempo de resposta torna-se mais ou menos RTT mais tempo de processamento, pois RTT_{FE} é insignificativamente pequeno e RTT_{BE} é mais ou menos RTT. Resumindo, a divisão do TCP pode reduzir o atraso da rede de cerca de 4 · RTT para RTT, melhorando significativamente o desempenho percebido pelo usuário, em particular para usuários que estão longe do centro de dados mais próximo. A divisão do TCP também ajuda a reduzir os atrasos de retransmissão do TCP causados por perdas nas redes de acesso. Hoje, Google e Akamai utilizam bastante seus servidores CDN em redes de acesso (ver Seção 7.2) para realizar a divisão do TCP para os serviços de nuvem que eles suportam [Chen, 2011].

Recuperação rápida

Na recuperação rápida, o valor de `cwnd` é aumentado em 1 MSS para cada ACK duplicado recebido no segmento perdido que fez o TCP entrar no modo de recuperação rápida. Mais cedo ou mais tarde, quando um ACK chega ao segmento perdido, o TCP entra no modo de prevenção de congestionamento após reduzir `cwnd`. Se houver um evento de esgotamento de temporização, a recuperação rápida é alterada para o modo de partida lenta após desempenhar as mesmas ações que a partida lenta e a prevenção de congestionamento: o valor de `cwnd` é ajustado para 1 MSS, e o valor de `ssthresh`, para metade do valor de `cwnd` no momento em que o evento de perda ocorreu.

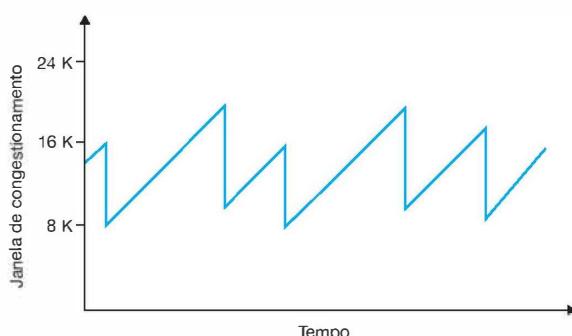
A recuperação rápida é recomendada, mas não exigida, para o protocolo TCP [RFC 5681]. É interessante o fato de que uma antiga versão do TCP, conhecida como **TCP Tahoe**, reduzia incondicionalmente sua janela de congestionamento para 1 MSS e entrava na fase de partida lenta após um evento de perda de esgotamento do temporizador ou de ACK duplicado triplo. A versão atual do TCP, a **TCP Reno**, incluiu a recuperação rápida.

A Figura 3.53 ilustra a evolução da janela de congestionamento do TCP para as versões Reno e Tahoe. Nessa figura, o limiar é, no início, igual a 8 MSS. Nas primeiras oito sessões de transmissão, as duas versões possuem ações idênticas. A janela de congestionamento se eleva exponencialmente rápido durante a partida lenta e atinge o limiar na quarta sessão de transmissão. A janela de congestionamento, então, se eleva de modo linear até que ocorra um evento ACK duplicado triplo, logo após a oitava sessão de transmissão. Observe que a janela de congestionamento é 12 • *MSS* quando ocorre o evento de perda. O valor de `ssthresh` é, então, ajustado para 0,5 • `cwnd` = 6 • MSS. No TCP Reno, a janela é ajustada para `cwnd` = 6 • MSS, e depois cresce linearmente. No TCP Tahoe, é ajustada para 1 MSS e cresce de modo exponencial até que alcance o valor de `ssthresh`, quando começa a crescer linearmente.

A Figura 3.52 apresentou a descrição FSM completa dos algoritmos de controle de congestionamento — partida lenta, prevenção de congestionamento e recuperação rápida. A figura também indica onde pode ocorrer transmissão de novos segmentos ou segmentos retransmitidos. Embora seja importante diferenciar controle/retransmissão de erro TCP de controle de congestionamento no TCP, também é importante avaliar como esses dois aspectos do TCP estão inseparavelmente ligados.

Controle de congestionamento no TCP: retrospectiva

Após nos aprofundarmos em detalhes sobre partida lenta, prevenção de congestionamento e recuperação rápida, vale a pena agora voltar e ver a floresta por entre as árvores. Desconsiderando o período inicial de partida lenta, quando uma conexão se inicia, e supondo que as perdas são indicadas por ACKs duplicados triplos e não por esgotamentos de temporização, o controle de congestionamento no TCP consiste em um aumento linear

FIGURA 3.53 EVOLUÇÃO DA JANELA DE CONGESTIONAMENTO DO TCP (TAHOE E RENO)

(aditivo) em cwnd de 1 MMS por RTT e, então, uma redução à metade (diminuição multiplicativa) de cwnd em um evento ACK duplicado triplo. Por esta razão, o controle de congestionamento no TCP é quase sempre denominado **aumento aditivo, diminuição multiplicativa (AIMD** — *Additive-Increase, Multiplicative-Decrease*). O controle de congestionamento AIMD faz surgir o comportamento semelhante a "dentes de serra", mostrado na Figura 3.54, a qual também ilustra de forma interessante nossa intuição anterior sobre a "sondagem" do TCP por largura de banda — o TCP aumenta linearmente o tamanho de sua janela de congestionamento (e, portanto, sua taxa de transmissão) até que ocorra um evento ACK duplicado triplo. Então, ele reduz o tamanho de sua janela por um fator de dois, mas começa de novo a aumentá-la linearmente, buscando saber se há uma largura de banda adicional disponível.

Como já foi observado, a maioria das implementações TCP, hoje, utiliza o algoritmo Reno [Padhye, 2001]. Foram propostas muitas variações desse algoritmo [RFC 3782; RFC 2018]. O algoritmo Vegas [Brakmo, 1995; Ahn, 1995] tenta evitar o congestionamento enquanto mantém uma boa vazão. A ideia básica de tal algoritmo é (1) detectar congestionamento nos roteadores entre a origem e o destino *antes* que ocorra a perda de pacote e (2) diminuir linearmente a taxa ao ser detectada essa perda de pacote iminente, que pode ser prevista por meio da observação do RTT. Quanto maior for o RTT dos pacotes, maior será o congestionamento nos roteadores. O Linux suporta diversos algoritmos de controle de congestionamento (incluindo o TCP Reno e o TCP Vegas) e permite que um administrador de sistema configure qual versão do TCP será utilizada. A versão padrão do TCP no Linux versão 2.6.18 foi definida para CUBIC [Ha, 2008], uma versão do TCP desenvolvida para aplicações de alta largura de banda. Para obter um estudo recente das muitas versões do TCP, consulte Afanasyev [2010].

O algoritmo AIMD do TCP foi desenvolvido com base em um grande trabalho de engenharia e experiências com controle de congestionamento em redes operacionais. Dez anos após o desenvolvimento do TCP, a análise teórica mostrou que o algoritmo de controle de congestionamento do TCP serve como um algoritmo de otimização assíncrona distribuída, que resulta em vários aspectos importantes do desempenho do usuário e da rede sendo otimizados em simultâneo [Kelly, 1998]. Uma rica teoria de controle de congestionamento foi desenvolvida desde então [Srikant, 2004].

Descrição macroscópica da vazão do TCP

Dado o comportamento de dentes de serra do TCP, é natural considerar qual seria a vazão média (isto é, a taxa média) de uma conexão TCP de longa duração. Nessa análise, vamos ignorar as fases de partida lenta que ocorrem após eventos de esgotamento de temporização. (Elas em geral são muito curtas, visto que o remetente sai com rapidez exponencial.) Durante determinado intervalo de viagem de ida e volta, a taxa à qual o TCP envia dados é uma função da janela de congestionamento e do *RTT* corrente. Quando o tamanho da janela for *w* bytes, e o tempo de viagem de ida e volta for *RTT* segundos, a taxa de transmissão do TCP será mais ou menos *w/RTT*. Então, o TCP faz

FIGURA 3.54 CONTROLE DE CONGESTIONAMENTO POR AUMENTO AUDITIVO, DIMINUIÇÃO MULTIPLICATIVA

uma sondagem em busca de alguma largura de banda adicional aumentando w em 1 MSS a cada RTT até ocorrer um evento de perda. Seja W o valor de w quando ocorre um evento de perda. Admitindo que RTT e W são mais ou menos constantes no período da conexão, a taxa de transmissão fica na faixa de $W/(2 \cdot RTT)$ a W/RTT.

Essas suposições levam a um modelo macroscópico muito simplificado para o comportamento do TCP em estado constante. A rede descarta um pacote da conexão quando a taxa aumenta para W/RTT; então a taxa é reduzida à metade e, em seguida, aumenta em MSS/RTT a cada RTT até alcançar W/RTT novamente. Esse processo se repete de modo contínuo. Como a vazão do TCP (isto é, sua taxa) aumenta linearmente entre os dois valores extremos, temos:

$$\text{vazão média de uma conexão} = \frac{0{,}75 \cdot W}{RTT}$$

Usando esse modelo muito idealizado para a dinâmica de regime permanente do TCP, podemos também derivar uma interessante expressão que relaciona a taxa de perda de uma conexão com sua largura de banda disponível [Mahdavi, 1997]. Essa derivação está delineada nos exercícios de fixação. Um modelo mais sofisticado que demonstrou empiricamente estar de acordo com dados medidos é apresentado em Padhye [2000].

TCP por caminhos com alta largura de banda

É importante perceber que o controle de congestionamento no TCP evoluiu ao longo dos anos e, na verdade, continua a evoluir. Um resumo das variantes atuais do TCP e uma discussão sobre a evolução do TCP podem ser vistos em Floyd (2001), RFC 5681 e Afanasyev (2010). O que era bom para a Internet quando a maior parte das conexões TCP carregava tráfego SMTP, FTP e Telnet nem sempre é bom para a de hoje, dominada pelo HTTP, ou para uma Internet futura, com serviços que ainda nem sonhamos.

A necessidade da evolução contínua do TCP pode ser ilustrada considerando as conexões TCP de alta velocidade que são necessárias para aplicações de computação em grade e nuvem. Por exemplo, suponha uma conexão TCP com segmentos de 1.500 bytes e RTT de 100 ms, e suponha que queremos enviar dados por essa conexão a 10 Gbits/s. Seguindo o [RFC 3649] e utilizando a fórmula de vazão do TCP apresentada anteriormente, notamos que, para alcançar uma vazão de 10 Gbits/s, o tamanho médio da janela de congestionamento precisaria ser 83.333 segmentos. São *muitos* segmentos, e pensar que um deles poderia ser perdido em trânsito nos deixa bastante preocupados. O que aconteceria no caso de uma perda? Ou, em outras palavras, que fração dos segmentos transmitidos poderia ser perdida e ainda assim permitir que o algoritmo de controle de congestionamento no TCP delineado na Tabela 3.52 alcançasse a taxa desejada de 10 Gbits/s? Nos exercícios de fixação deste capítulo você acompanhará a dedução de uma fórmula que relaciona a vazão de uma conexão TCP em função da taxa de perda (L), do tempo de viagem de ida e volta (RTT) e do tamanho máximo de segmento (MSS):

$$\text{vazão média de uma conexão} = \frac{1{,}22 \cdot MSS}{RTT\sqrt{L}}$$

Usando essa fórmula, podemos ver que, para alcançar uma vazão de 10 Gbits/s, o algoritmo de controle de congestionamento no TCP de hoje pode tolerar uma probabilidade de perda de segmentos de apenas $2 \cdot 10^{-10}$ (equivalente a um evento de perda a cada 5 bilhões de segmentos) — uma taxa muito baixa. Essa observação levou muitos pesquisadores a investigar novas versões do TCP projetadas para esses ambientes de alta velocidade; Jin (2004); RFC 3649; Kelly (2003); Ha (2008) apresentam discussões sobre esses esforços.

3.7.1 Equidade

Considere K conexões TCP, cada uma com um caminho fim a fim diferente, mas todas passando pelo gargalo em um enlace com taxa de transmissão de R bits/s (aqui, *gargalo em um enlace* quer dizer que nenhum

dos outros enlaces ao longo do caminho de cada conexão está congestionado e que todos dispõem de abundante capacidade de transmissão em comparação à capacidade de transmissão do enlace com gargalo). Suponha que cada conexão está transferindo um arquivo grande e que não há tráfego UDP passando pelo enlace com gargalo. Dizemos que um mecanismo de controle de congestionamento é *justo* se a taxa média de transmissão de cada conexão for mais ou menos R/K; isto é, cada uma obtém uma parcela igual da largura de banda do enlace.

O algoritmo AIMD do TCP é justo, considerando, em especial, que diferentes conexões TCP podem começar em momentos diferentes e, assim, ter tamanhos de janela diferentes em um dado instante? Chiu [1989] explica, de um modo elegante e intuitivo, por que o controle de congestionamento converge para fornecer um compartilhamento justo da largura de banda do enlace entre conexões TCP concorrentes.

Vamos considerar o caso simples de duas conexões TCP compartilhando um único enlace com taxa de transmissão R, conforme mostra a Figura 3.55. Admitamos que as duas conexões tenham os mesmos MSS e RTT (de modo que, se o tamanho de suas janelas de congestionamento for o mesmo, eles terão a mesma vazão) e uma grande quantidade de dados para enviar e que nenhuma outra conexão TCP ou datagramas UDP atravesse esse enlace compartilhado. Vamos ignorar também a fase de partida lenta do TCP e admitir que as conexões TCP estão operando em modo prevenção de congestionamento (AIMD) todo o tempo.

A Figura 3.56 mostra a vazão alcançada pelas duas conexões TCP. Se for para o TCP compartilhar equitativamente a largura de banda do enlace entre as duas conexões, a vazão alcançada deverá cair ao longo da linha a 45 graus (igual compartilhamento da largura de banda) que parte da origem. Idealmente, a soma das duas vazões seria igual a R. (Com certeza, não é uma situação desejável cada conexão receber um compartilhamento igual, mas igual a zero, da capacidade do enlace!) Portanto, o objetivo é que as vazões alcançadas fiquem em algum lugar perto da intersecção da linha de "igual compartilhamento da largura de banda" com a linha de "utilização total da largura de banda" da Figura 3.56.

Suponha que os tamanhos de janela TCP sejam tais que, em um determinado instante, as conexões 1 e 2 alcancem as vazões indicadas pelo ponto A na Figura 3.56. Como a quantidade de largura de banda do enlace consumida em conjunto pelas duas conexões é menor do que R, não ocorrerá nenhuma perda e ambas as conexões aumentarão suas janelas em 1 MSS por RTT como resultado do algoritmo de prevenção de congestionamento do TCP. Assim, a vazão conjunta das duas conexões continua ao longo da linha a 45 graus (aumento igual para as duas), começando no ponto A. Por fim, a largura de banda do enlace consumida em conjunto pelas duas conexões será maior do que R e, assim, por fim ocorrerá perda de pacote. Suponha que as conexões 1 e 2 experimentem perda de pacote quando alcançarem as vazões indicadas pelo ponto B. As conexões 1 e 2 então reduzirão suas janelas por um fator de 2. Assim, as vazões resultantes são as do ponto C, a meio caminho do vetor que começa em B e termina na origem. Como a utilização conjunta da largura de banda é menor do que R no ponto C, as duas conexões novamente aumentam suas vazões ao longo da linha a 45 graus que começa no ponto C. Mais cedo ou mais tarde ocorrerá perda, por exemplo, no ponto D, e as duas conexões reduzirão de novo o tamanho de suas janelas por um fator de 2 — e assim por diante. Você pode ter certeza de que a largura de banda alcançada pelas duas conexões flutuará ao longo da linha de igual compartilhamento da largura de banda. E também estar certo de que as duas conexões convergirão para esse comportamento, não importando onde elas comecem no

FIGURA 3.55 DUAS CONEXÕES TCP COMPARTILHANDO UM ÚNICO ENLACE CONGESTIONADO

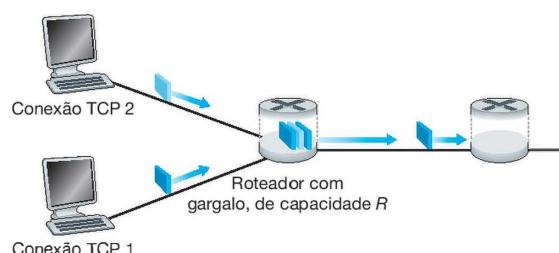
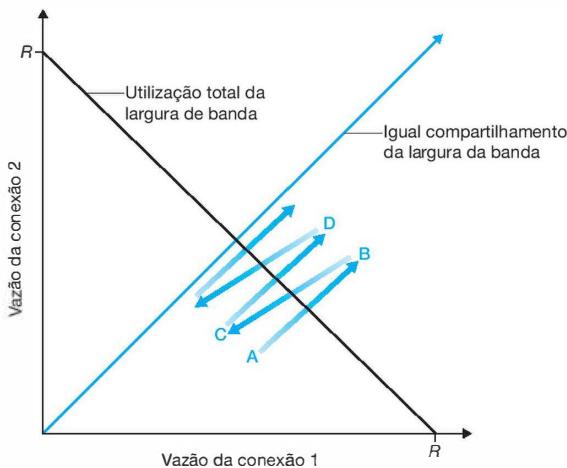



FIGURA 3.56 VAZÃO ALCANÇADA PELAS CONEXÕES TCP 1 E TCP 2

espaço bidimensional! Embora haja uma série de suposições idealizadas por trás desse cenário, ainda assim ele dá uma ideia intuitiva de por que o TCP resulta em igual compartilhamento da largura de banda entre conexões.

Em nosso cenário idealizado, admitimos que apenas conexões TCP atravessem o enlace com gargalo, que elas tenham o mesmo valor de RTT e que uma única conexão TCP esteja associada com um par hospedeiro/destinatário. Na prática, essas condições não são muito encontradas e, assim, é possível que as aplicações cliente-servidor obtenham porções muito desiguais da largura de banda do enlace. Em especial, foi demonstrado que, quando várias conexões compartilham um único enlace com gargalo, as sessões cujos RTTs são menores conseguem obter a largura de banda disponível naquele enlace mais rapidamente (isto é, abre suas janelas de congestionamento mais depressa) à medida que o enlace fica livre. Assim, conseguem vazões mais altas do que conexões com RTTs maiores [Lakshman, 1997].

Equidade e UDP

Acabamos de ver como o controle de congestionamento no TCP regula a taxa de transmissão de uma aplicação por meio do mecanismo de janela de congestionamento. Diversas aplicações de multimídia, como telefone por Internet e videoconferência, muitas vezes não rodam sobre TCP exatamente por essa razão — elas não querem que sua taxa de transmissão seja limitada, mesmo que a rede esteja muito congestionada. Ao contrário, preferem rodar sobre UDP, que não tem controle de congestionamento. Quando rodam sobre esse protocolo, as aplicações podem passar seus áudios e vídeos para a rede a uma taxa constante e, de modo ocasional, perder pacotes, em vez de reduzir suas taxas a níveis "justos" em horários de congestionamento e não perder nenhum deles. Do ponto de vista do TCP, as aplicações de multimídia que rodam sobre UDP não são justas — elas não cooperam com as outras conexões nem ajustam suas taxas de transmissão de maneira adequada. Como o controle de congestionamento no TCP reduzirá sua taxa de transmissão quando houver aumento de congestionamento (perda), enquanto origens UDP não precisam fazer o mesmo, é possível que essas origens desalojem o tráfego TCP. Uma área importante da pesquisa hoje é o desenvolvimento de mecanismos de controle que impeçam que o tráfego de UDP leve a vazão da Internet a uma parada repentina [Floyd, 1999; Floyd, 2000; Kohler, 2006].

Equidade e conexões TCP paralelas

Mas, mesmo que pudéssemos obrigar o tráfego UDP a se comportar com equidade, o problema ainda não estaria resolvido por completo. Isso porque não há nada que impeça uma aplicação de rodar sobre TCP usando múltiplas conexões paralelas. Por exemplo, navegadores Web com frequência usam múltiplas conexões TCP paralelas para transferir os vários objetos de uma página. (O número exato de conexões múltiplas pode ser configurado na maioria dos navegadores.) Quando usa múltiplas conexões paralelas, uma aplicação consegue uma fração maior da largura de banda de um enlace congestionado. Como exemplo, considere um enlace de taxa R que está suportando nove aplicações cliente-servidor em curso, e cada uma das aplicações está usando uma conexão TCP. Se surgir uma nova aplicação que também utilize uma conexão TCP, então cada aplicação conseguirá aproximadamente a mesma taxa de transmissão igual a $R/10$. Porém, se, em vez disso, essa nova aplicação usar 11 conexões TCP paralelas, então ela conseguirá uma alocação injusta de mais do que $R/2$. Como a penetração do tráfego Web na Internet é grande, as múltiplas conexões paralelas não são incomuns.

3.8 RESUMO

Começamos este capítulo estudando os serviços que um protocolo de camada de transporte pode prover às aplicações de rede. Por um lado, esse protocolo pode ser muito simples e oferecer serviços básicos às aplicações, provendo apenas uma função de multiplexação/demultiplexação para processos em comunicação. O protocolo UDP da Internet é um exemplo desse serviço básico. Por outro lado, pode fornecer uma série de garantias às aplicações, como entrega confiável de dados, garantias contra atrasos e garantias de largura de banda. Não obstante, os serviços que um protocolo de transporte pode prover são muitas vezes limitados pelo modelo de serviço do protocolo subjacente de camada de rede. Se o protocolo de camada de rede não puder proporcionar garantias contra atraso ou garantias de largura de banda para segmentos da camada de transporte, então o protocolo de camada de transporte não poderá fornecer tais garantias para as mensagens enviadas entre processos.

Aprendemos na Seção 3.4 que um protocolo de camada de transporte pode prover transferência confiável de dados mesmo que a camada de rede subjacente seja não confiável. Vimos que há muitos pontos sutis na transferência confiável de dados, mas que a tarefa pode ser realizada pela combinação cuidadosa de reconhecimentos, temporizadores, retransmissões e números de sequência.

Embora tenhamos examinado a transferência confiável de dados neste capítulo, devemos ter em mente que ela pode ser fornecida por protocolos de camada de enlace, de rede, de transporte ou de aplicação. Qualquer uma das camadas superiores da pilha de protocolos pode executar reconhecimentos, temporizadores, retransmissões e números de sequência e prover transferência confiável de dados para a camada situada acima dela. Na verdade, com o passar dos anos, engenheiros e cientistas da computação projetaram e realizaram, independentemente, protocolos de camada de enlace, de rede, de transporte e de aplicação que fornecem transferência confiável de dados (embora muitos tenham desaparecido silenciosamente).

Na Seção 3.5, examinamos em detalhes o TCP, o protocolo de camada de transporte confiável orientado para conexão da Internet. Aprendemos que o TCP é complexo e envolve conexão, controle de fluxo, estimativa de tempo de viagem de ida e volta, bem como transferência confiável de dados. Na verdade, o TCP é mais complexo do que nossa descrição — de propósito, não discutimos uma série de ajustes, acertos e melhorias que estão implementados em várias versões do TCP. Toda essa complexidade, no entanto, fica escondida da aplicação de rede. Se um cliente em um hospedeiro quiser enviar dados de maneira confiável para outro hospedeiro, ele apenas abre um *socket* TCP para o servidor e passa dados para dentro desse *socket*. A aplicação cliente-servidor felizmente fica alheia a toda a complexidade do TCP.

Na Seção 3.6, examinamos o controle de congestionamento de um ponto de vista mais amplo e, na Seção 3.7, demonstramos como o TCP realiza controle de congestionamento. Aprendemos que esse controle é imperativo para o bem-estar da rede. Sem ele, uma rede pode facilmente ficar travada, com pouco ou nenhum dado sendo transportado fim a fim. Na Seção 3.7, aprendemos também que o TCP executa um mecanismo de controle de

congestionamento fim a fim que aumenta aditivamente sua taxa de transmissão quando julga que o caminho da conexão TCP está livre de congestionamento e reduz multiplicativamente sua taxa de transmissão quando ocorre perda. Esse mecanismo também luta para dar a cada conexão TCP que passa por um enlace congestionado uma parcela igual da largura de banda da conexão. Examinamos ainda com algum detalhe o impacto do estabelecimento da conexão TCP e da partida lenta sobre a latência. Observamos que, em muitos cenários importantes, o estabelecimento da conexão e a partida lenta contribuem de modo significativo para o atraso fim a fim. Enfatizamos mais uma vez que, embora tenha evoluído com o passar dos anos, o controle de congestionamento no TCP permanece como uma área de pesquisa intensa e, provavelmente, continuará a evoluir nos anos vindouros.

Nossa discussão sobre os protocolos específicos de transporte da Internet neste capítulo concentrou-se no UDP e no TCP — os dois "burros de carga" da camada de transporte. Entretanto, duas décadas de experiência com os dois identificaram circunstâncias nas quais nenhum deles é apropriado de maneira ideal. Desse modo, pesquisadores se ocuparam em desenvolver protocolos da camada de transporte adicionais, muitos dos quais são, agora, padrões propostos pelo IETF.

O Protocolo de Controle de Congestionamento de Datagrama (DCCP) [RFC 4340] oferece um serviço de baixo consumo, orientado a mensagem e não confiável de forma semelhante ao UDP, mas com uma forma selecionada de aplicação de controle de congestionamento que é compatível com o TCP. Caso uma aplicação necessitar de uma transferência de dados confiável ou semiconfiável, então isso seria realizado dentro da própria aplicação, talvez utilizando os mecanismos que estudamos na Seção 3.4. O DCCP é utilizado em aplicações, como o fluxo de mídia (veja Capítulo 7), que podem se aproveitar da escolha entre conveniência e confiança do fornecimento de dados, mas que querem ser sensíveis ao congestionamento da rede.

O Protocolo de Controle de Fluxo de Transmissão (SCTP) [RFC 4960, RFC 3286] é um protocolo confiável e orientado a mensagens que permite que diferentes "fluxos" de aplicação sejam multiplexados por meio de uma única conexão SCTP (método conhecido como "múltiplos fluxos"). De um ponto de vista confiável, os diferentes fluxos dentro da conexão são controlados em separado, de modo que uma perda de pacote em um fluxo não afete o fornecimento de dados em outros fluxos. O SCTP também permite que os dados sejam transferidos por meio de dois percursos de saída quando um hospedeiro está conectado a duas ou mais redes, fornecimento opcional de dados fora de ordem, e muitos outros recursos. Os algoritmos de controle de congestionamento e de fluxo do SCTP são basicamente os mesmos do TCP.

O protocolo TFRC (*TCP — Friendly Rate Control*) [RFC 5348] é mais um protocolo de controle de congestionamento do que um protocolo da camada de transporte completo. Ele especifica um mecanismo de controle que poderia ser utilizado em outro protocolo de transporte como o DCCP (de fato, um dos dois protocolos de aplicação, disponível no DCCP, é o TFRC). O objetivo do TFRC é estabilizar o comportamento do tipo "dente de serra" (veja Figura 3.54) no controle de congestionamento no TCP, enquanto mantém uma taxa de envio de longo prazo "razoavelmente" próxima à do TCP. Com uma taxa de envio mais estável do que a do TCP, o TFRC é adequado às aplicações multimídia, como telefonia IP ou fluxo de mídia, em que uma taxa estável é importante. O TFRC é um protocolo baseado em equação que utiliza a taxa de perda de pacote calculada como suporte a uma equação que estima qual seria a vazão do TCP se uma sessão TCP sofrer taxa de perda [Padhye, 2000]. Essa taxa é então adotada como a taxa-alvo de envio do TFRC.

Somente o futuro dirá se o DCCP, o SCTP ou o TFRC possuirá uma implementação difundida. Enquanto esses protocolos fornecem claramente recursos avançados sobre o TCP e o UDP, estes já provaram a si mesmos ser "bons o bastante" com o passar dos anos. A vitória do "melhor" sobre o "bom o bastante" dependerá de uma mistura complexa de considerações técnicas, sociais e comerciais.

No Capítulo 1, dissemos que uma rede de computadores pode ser dividida em "periferia da rede" e "núcleo da rede". A periferia abrange tudo o que acontece nos sistemas finais. Com o exame da camada de aplicação e da camada de transporte, nossa discussão sobre a periferia da rede está completa. É hora de explorar o núcleo! Essa jornada começa no próximo capítulo, em que estudaremos a camada de rede, e continua no Capítulo 5, em que estudaremos a camada de enlace.

EXERCÍCIOS DE FIXAÇÃO E PERGUNTAS

Questões de revisão do Capítulo 3

SEÇÕES 3.1–3.3

R1. Suponha que uma camada de rede forneça o seguinte serviço. A camada de rede no computador-fonte aceita um segmento de tamanho máximo de 1.200 bytes e um endereço de computador-alvo da camada de transporte. A camada de rede, então, garante encaminhar o segmento para a camada de transporte no computador-alvo. Suponha que muitos processos de aplicação de rede possam estar sendo executados no hospedeiro de destino.

a. Crie, da forma mais simples, o protocolo da camada de transporte possível que levará os dados da aplicação para o processo desejado no hospedeiro de destino. Suponha que o sistema operacional do hospedeiro de destino determinou um número de porta de 4 bytes para cada processo de aplicação em execução.

b. Modifique esse protocolo para que ele forneça um "endereço de retorno" ao processo-alvo.

c. Em seus protocolos, a camada de transporte "tem de fazer algo" no núcleo da rede de computadores?

R2. Considere um planeta onde todos possuam uma família com seis membros, cada família viva em sua própria casa, cada casa possua um endereço único e cada pessoa em certa casa possua um único nome. Suponha que esse planeta possua um serviço postal que entregue cartas da casa-fonte à casa-alvo. O serviço exige que (1) a carta esteja em um envelope e que (2) o endereço da casa-alvo (e nada mais) esteja escrito claramente no envelope. Imagine que cada família possua um membro representante que recebe e distribui cartas para as demais. As cartas não apresentam necessariamente qualquer indicação dos destinatários das cartas.

a. Utilizando a solução do Problema R1 como inspiração, descreva um protocolo que os representantes possam utilizar para entregar cartas de um membro remetente de uma família para um membro destinatário de outra família.

b. Em seu protocolo, o serviço postal precisa abrir o envelope e verificar a carta para fornecer o serviço?

R3. Considere uma conexão TCP entre o hospedeiro A e o hospedeiro B. Suponha que os segmentos TCP que trafegam do hospedeiro A para o hospedeiro B tenham número de porta da origem x e número de porta do destino y. Quais são os números de porta da origem e do destino para os segmentos que trafegam do hospedeiro B para o hospedeiro A?

R4. Descreva por que um desenvolvedor de aplicação pode escolher rodar uma aplicação sobre UDP em vez de sobre TCP.

R5. Por que o tráfego de voz e de vídeo é frequentemente enviado por meio do UDP e não do TCP na Internet de hoje? (*Dica:* A resposta que procuramos não tem nenhuma relação com o mecanismo de controle de congestionamento no TCP.)

R6. É possível que uma aplicação desfrute de transferência confiável de dados mesmo quando roda sobre UDP? Caso a resposta seja afirmativa, como isso acontece?

R7. Suponha que um processo no hospedeiro C possua um *socket* UDP com número de porta 6789 e que o hospedeiro A e o hospedeiro B, individualmente, enviem um segmento UDP ao hospedeiro C com número de porta de destino 6789. Os dois segmentos serão encaminhados para o mesmo *socket* no hospedeiro C? Se sim, como o processo no hospedeiro C saberá que os dois segmentos vieram de dois hospedeiros diferentes?

R8. Suponha que um servidor da Web seja executado no computador C na porta 80. Esse servidor utiliza conexões contínuas e, no momento, está recebendo solicitações de dois computadores diferentes, A e B. Todas as solicitações estão sendo enviadas por meio do mesmo *socket* no computador C? Se estão passando por diferentes *sockets*, dois deles possuem porta 80? Discuta e explique.

SEÇÃO 3.4

R9. Em nossos protocolos `rdt`, por que precisamos introduzir números de sequência?

R10. Em nossos protocolos `rdt`, por que precisamos introduzir temporizadores?

R11. Suponha que o atraso de viagem de ida e volta entre o emissor e o receptor seja constante e conhecido para o emissor. Ainda seria necessário um temporizador no protocolo `rdt 3.0`, supondo que os pacotes podem ser perdidos? Explique.

R12. Visite o applet Go-Back-N Java no site de apoio do livro.

a. A origem enviou cinco pacotes e depois interrompeu a animação antes que qualquer um dos cinco pacotes chegasse ao destino. Então, elimine o primeiro pacote e reinicie a animação. Descreva o que acontece.

b. Repita o experimento, mas agora deixe o primeiro pacote chegar ao destino e elimine o primeiro reconhecimento. Descreva novamente o que acontece.

c. Por fim, tente enviar seis pacotes. O que acontece?

R13. Repita a Questão 12, mas agora com o applet Java Selective Repeat. O que difere o Selective Repeat do Go-Back-N?

SEÇÃO 3.5

R14. Falso ou verdadeiro?

a. O hospedeiro A está enviando ao hospedeiro B um arquivo grande por uma conexão TCP. Suponha que o hospedeiro B não tenha dados para enviar para o hospedeiro A. O hospedeiro B não enviará reconhecimentos para A porque ele não pode dar carona aos reconhecimentos dos dados.

b. O tamanho de `rwnd` do TCP nunca muda enquanto dura a conexão.

c. Suponha que o hospedeiro A esteja enviando ao hospedeiro B um arquivo grande por uma conexão TCP. O número de bytes não reconhecidos que o hospedeiro A envia não pode exceder o tamanho do buffer de recepção.

d. Imagine que o hospedeiro A esteja enviando ao hospedeiro B um arquivo grande por uma conexão TCP. Se o número de sequência para um segmento dessa conexão for m, então o número de sequência para o segmento subsequente será necessariamente $m + 1$.

e. O segmento TCP tem um campo em seu cabeçalho para `rwnd`.

f. Suponha que o último `SampleRTT` de uma conexão TCP seja igual a 1 s. Então, o valor corrente de `TimeoutInterval` para a conexão será necessariamente ajustado para um valor ≥ 1 s.

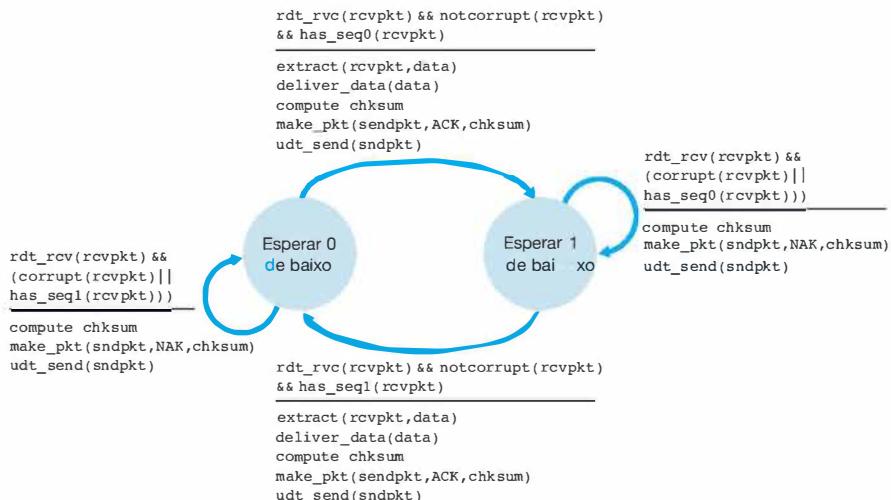
g. Imagine que o hospedeiro A envie ao hospedeiro B, por uma conexão TCP, um segmento com o número de sequência 38 e 4 bytes de dados. Nesse mesmo segmento, o número de reconhecimento será necessariamente 42.

R15. Suponha que o hospedeiro A envie dois segmentos TCP um atrás do outro ao hospedeiro B sobre uma conexão TCP. O primeiro segmento tem número de sequência 90 e o segundo, número de sequência 110.

a. Quantos dados tem o primeiro segmento?

b. Suponha que o primeiro segmento seja perdido, mas o segundo chegue a B. No reconhecimento que B envia a A, qual será o número de reconhecimento?

R16. Considere o exemplo do Telnet discutido na Seção 3.5. Alguns segundos após o usuário digitar a letra "C", ele digitará a letra "R". Depois disso, quantos segmentos serão enviados e o que será colocado nos campos de número de sequência e de reconhecimento dos segmentos?

SEÇÃO 3.7

R17. Suponha que duas conexões TCP estejam presentes em algum enlace congestionado de velocidade R bits/s. As duas conexões têm um arquivo imenso para enviar (na mesma direção, pelo enlace congestionado).

As transmissões dos arquivos começam exatamente ao mesmo tempo. Qual é a velocidade de transmissão que o TCP gostaria de dar a cada uma das conexões?

R18. Verdadeiro ou falso: considere o controle de congestionamento no TCP. Quando um temporizador expira no remetente, o valor de `ssthresh` é ajustado para a metade de seu valor anterior.

R19. Na discussão sobre divisão do TCP, na nota em destaque da Seção 3.7, afirmamos que o tempo de resposta com a divisão do TCP é aproximadamente $4 \cdot RTT_{FE} + RTT_{BE}$ + tempo de processamento. Justifique essa afirmação.

PROBLEMAS

P1. Suponha que o cliente A inicie uma sessão Telnet com o servidor S. Quase ao mesmo tempo, o cliente B também inicia uma sessão Telnet com o servidor S. Forneça possíveis números de porta da fonte e do destino para:

a. Os segmentos enviados de A para S.

b. Os segmentos enviados de B para S.

c. Os segmentos enviados de S para A.

d. Os segmentos enviados de A para B.

e. Se A e B são hospedeiros diferentes, é possível que o número de porta da fonte nos segmentos de A para S seja o mesmo que nos de B para S?

f. E se forem o mesmo hospedeiro?

P2. Considere a Figura 3.5. Quais são os valores da porta de fonte e da porta de destino nos segmentos que fluem do servidor de volta aos processos clientes? Quais são os endereços IP nos datagramas de camada de rede que carregam os segmentos de camada de transporte?

P3. O UDP e o TCP usam complementos de 1 para suas somas de verificação. Suponha que você tenha as seguintes três palavras de 8 bits: 01010011, 01100110 e 01110100. Qual é o complemento de 1 para as somas dessas palavras? (Note que, embora o UDP e o TCP usem palavras de 16 bits no cálculo da soma de verificação, nesse problema solicitamos que você considere somas de 8 bits.) Mostre todo o trabalho. Por que o UDP toma o complemento de 1 da soma, isto é, por que não toma apenas a soma? Com o esquema de complemento de 1, como o destinatário detecta erros? É possível que um erro de 1 bit passe despercebido? E um erro de 2 bits?

P4 a. Suponha que você tenha os seguintes bytes: 01011100 e 01100101. Qual é o complemento de 1 da soma desses 2 bytes?

b. Suponha que você tenha os seguintes bytes: 11011010 e 01100101. Qual é o complemento de 1 da soma desses 2 bytes?

c. Para os bytes do item (a), dê um exemplo em que um bit é invertido em cada um dos 2 bytes e, mesmo assim, o complemento de um não muda.

P5. Suponha que o receptor UDP calcule a soma de verificação da Internet para o segmento UDP recebido e encontre que essa soma coincide com o valor transportado no campo da soma de verificação. O receptor pode estar absolutamente certo de que não ocorreu nenhum erro de bit? Explique.

P6. Considere nosso motivo para corrigir o protocolo `rtd2.1`. Demonstre que o destinatário apresentado na Figura 3.57, quando em operação com o remetente mostrado na Figura 3.11, pode levar remetente e destinatário a entrar em estado de travamento, em que cada um espera por um evento que nunca vai ocorrer.

P7. No protocolo `rdt3.0`, os pacotes ACK que fluem do destinatário ao remetente não têm números de sequência (embora tenham um campo de ACK que contém o número de sequência do pacote que estão reconhecendo). Por que nossos pacotes ACK não requerem números de sequência?

P8. Elabore a FSM para o lado destinatário do protocolo `rdt3.0`.

FIGURA 3.57 UM RECEPTOR INCORRETO PARA O PROTOCOLO `rdt 2.1`.

P9. Elabore um diagrama de mensagens para a operação do protocolo `rdt3.0` quando pacotes de dados e de reconhecimento estão truncados. Seu diagrama deve ser semelhante ao usado na Figura 3.16.

P10. Considere um canal que pode perder pacotes, mas cujo atraso máximo é conhecido. Modifique o protocolo `rdt2.1` para incluir esgotamento de temporização do remetente e retransmissão. Informalmente, argumente por que seu protocolo pode se comunicar de modo correto por esse canal.

P11. Considere o `rdt2.2` destinatário da Figura 3.14 e a criação de um novo pacote na autotransição (isto é, a transição do estado de volta para si mesmo) nos estados "Esperar 0 de baixo" e "Esperar 1 de baixo": `sndpkt=make_pkt(ACK,0,checksum)` e `sndpkt=make_pkt(ACK,0, checksum)`. O protocolo funcionaria corretamente se essa ação fosse removida da autotransição no estado "Esperar 1 de baixo"? Justifique sua resposta. E se esse evento fosse removido da autotransição no estado "Esperar 0 de baixo"? [*Dica:* Neste último caso, considere o que aconteceria se o primeiro pacote do remetente ao destinatário fosse corrompido.]

P12. O lado remetente do `rdt3.0` simplesmente ignora (isto é, não realiza nenhuma ação) todos os pacotes recebidos que estão errados ou com o valor errado no campo `acknum` de um pacote de reconhecimento. Suponha que em tais circunstâncias o `rdt3.0` devesse apenas retransmitir o pacote de dados corrente. Nesse caso, o protocolo ainda funcionaria? (*Dica*: Considere o que aconteceria se houvesse apenas erros de bits; não há perdas de pacotes, mas podem ocorrer esgotamentos de temporização prematuros. Imagine quantas vezes o *enésimo* pacote é enviado, no limite em que *n* se aproximasse do infinito.)

P13. Considere o protocolo `rdt3.0`. Elabore um diagrama mostrando que, se a conexão de rede entre o remetente e o destinatário puder alterar a ordem de mensagens (isto é, se for possível reordenar duas mensagens que se propagam no meio entre o remetente e o destinatário), então o protocolo bit alternante não funcionará corretamente (lembre-se de identificar com clareza o sentido no qual o protocolo não funcionará de modo correto). Seu diagrama deve mostrar o remetente à esquerda e o destinatário à direita; o eixo do tempo deverá estar orientado de cima para baixo na página e mostrar a troca de mensagem de dados (D) e de reconhecimento (A). Não se esqueça de indicar o número de sequência associado com qualquer segmento de dados ou de reconhecimento.

P14. Considere um protocolo de transferência confiável de dados que use somente reconhecimentos negativos. Suponha que o remetente envie dados com pouca frequência. Um protocolo que utiliza somente NAKs seria preferível a um protocolo que utiliza ACKs? Por quê? Agora suponha que o remetente tenha uma grande

quantidade de dados para enviar e que a conexão fim a fim sofra poucas perdas. Nesse segundo caso, um protocolo que utilize somente NAKs seria preferível a um protocolo que utilize ACKs? Por quê?

P15. Considere o exemplo em que se atravessa os Estados Unidos mostrado na Figura 3.17. Que tamanho deveria ter a janela para que a utilização do canal fosse maior do que 98%? Suponha que o tamanho de um pacote seja 1.500 bytes, incluindo os campos do cabeçalho e os dados.

P16. Suponha que uma aplicação utilize `rdt3.0` como seu protocolo da camada de transporte. Como o protocolo pare e espere possui uma utilização do canal muito baixa (mostrada no exemplo de travessia dos Estados Unidos), os criadores dessa aplicação permitem que o receptor continue enviando de volta um número (mais do que dois) de ACK 0 alternado e ACK 1, mesmo que os dados correspondentes não cheguem ao receptor. O projeto dessa aplicação aumentaria a utilização do canal? Por quê? Há possíveis problemas com esse método? Explique.

P17. Considere duas entidades de rede, A e B, que estão conectadas por um canal bidirecional perfeito (isto é, qualquer mensagem enviada será recebida corretamente; o canal não corromperá, perderá nem reordenará pacotes). A e B devem entregar mensagens de dados entre si de forma alternada: primeiro, A deve entregar uma mensagem a B, depois B deve entregar uma mensagem a A, e assim por diante. Se uma entidade estiver em um estado onde não deve tentar entregar uma mensagem ao outro lado e houver um evento como a chamada `rdt_send(data)` de cima que tente transmitir dados para baixo, para o outro lado, essa chamada de cima pode apenas ser ignorada com uma chamada a `rdt_unable_to_send(data)`, que informa à camada de cima que atualmente não é possível enviar dados. [*Nota*: Essa suposição simplificada é para que você não tenha de se preocupar com o armazenamento de dados em buffer.]
Elabore uma especificação FSM para este protocolo (uma FSM para A e uma para B!). Observe que você não precisa se preocupar com um mecanismo de confiabilidade aqui; o ponto importante da questão é criar uma especificação FSM que reflita o comportamento sincronizado das duas entidades. Você deverá usar os seguintes eventos e ações que possuem o mesmo significado do protocolo `rdt1.0` da Figura 3.9: `rdt_send(data)`, `packet = make_pkt(data)`, `udt_send(packet)`, `rdt_rcv(packet)`, `extract(packet,data)`, `deliver_data(data)`. Cuide para que o protocolo reflita a alternância estrita de envio entre A e B. Além disso, não se esqueça de indicar os estados iniciais de A e B em suas especificações FSM.

P18. No protocolo genérico SR que estudamos na Seção 3.4.4, o remetente transmite uma mensagem assim que ela está disponível (se ela estiver na janela), sem esperar por um reconhecimento. Suponha, agora, que queiramos um protocolo SR que envie duas mensagens de cada vez. Isto é, o remetente enviará um par de mensagens, e o par de mensagens subsequente somente deverá ser enviado quando o remetente souber que ambas as mensagens do primeiro par foram recebidas corretamente.
Suponha que o canal possa perder mensagens, mas que não as corromperá nem as reordenará. Elabore um protocolo de controle de erro para a transferência confiável unidirecional de mensagens. Dê uma descrição FSM do remetente e do destinatário. Descreva o formato dos pacotes enviados entre o remetente e o destinatário e vice-versa. Se você usar quaisquer procedimentos de chamada que não sejam os da Seção 3.4 (por exemplo, `udt_send()`, `start_timer()`, `rdt_rcv()` etc.), esclareça as ações desses procedimentos. Dê um exemplo (um diagrama de mensagens para o remetente e para o destinatário) mostrando como seu protocolo se recupera de uma perda de pacote.

P19. Considere um cenário em que o hospedeiro A queira enviar pacotes para os hospedeiros B e C simultaneamente. O hospedeiro A está conectado a B e a C por um canal *broadcast* — um pacote enviado por A é levado pelo canal a B e a C. Suponha que o canal *broadcast* que conecta A, B e C possa, de modo independente, perder e corromper mensagens (e assim, por exemplo, uma mensagem enviada de A poderia ser recebida corretamente por B, mas não por C). Projete um protocolo de controle de erro do tipo pare e espere para a transferência confiável de um pacote de A para B e para C, tal que A não receba novos dados da camada superior até que saiba que B e C receberam corretamente o pacote em questão. Dê descrições FSM de A e C. (*Dica*: a FSM para B deve ser a mesma que para C.) Também dê uma descrição do(s) formato(s) de pacote usado(s).

P20. Considere um cenário em que os hospedeiros A e B queiram enviar mensagens ao hospedeiro C. Os hospedeiros A e C estão conectados por um canal que pode perder e corromper (e não reordenar) mensagens. B e C estão conectados por outro canal (independente do canal que conecta A e C) com as mesmas propriedades.

A camada de transporte no hospedeiro C deve alternar o envio de mensagens de A e B para a camada acima (ou seja, ela deve primeiro transmitir os dados de um pacote de A e depois os dados de um pacote de B, e assim por diante). Elabore um protocolo de controle de erro pare e espere para transferência confiável de pacotes de A e B para C, com envio alternado em C, como descrito acima. Dê descrições FSM de A e C. (*Dica:* a FSM para B deve ser basicamente a mesma de A.) Dê, também, uma descrição do(s) formato(s) de pacote utilizado(s).

P21. Suponha que haja duas entidades de rede A e B e que B tenha um suprimento de mensagens de dados que será enviado a A de acordo com as seguintes convenções: quando A recebe uma solicitação da camada superior para obter a mensagem de dados (D) seguinte de B, A deve enviar uma mensagem de requisição (R) para B no canal A-a-B; somente quando B receber uma mensagem R, ele poderá enviar uma mensagem de dados (D) de volta a A pelo canal B a A; A deve entregar uma cópia de cada mensagem D à camada superior; mensagens R podem ser perdidas (mas não corrompidas) no canal A-a-B; mensagens (D), uma vez enviadas, são sempre entregues corretamente; o atraso entre ambos os canais é desconhecido e variável.
Elabore um protocolo (dê uma descrição FSM) que incorpore os mecanismos apropriados para compensar a propensão à perda do canal A a B e implemente passagem de mensagem para a camada superior na entidade A, como discutido antes. Utilize apenas os mecanismos absolutamente necessários.

P22. Considere o protocolo GBN com um tamanho de janela remetente de 4 e uma faixa de números de sequência de 1.024. Suponha que, no tempo *t*, o pacote seguinte na ordem, pelo qual o destinatário está esperando, tenha um número de sequência *k*. Admita que o meio não reordene as mensagens. Responda às seguintes perguntas:

a. Quais são os possíveis conjuntos de números de sequência dentro da janela do remetente no tempo *t*? Justifique sua resposta.

b. Quais são todos os possíveis valores do campo ACK em todas as mensagens que estão atualmente se propagando de volta ao remetente no tempo *t*? Justifique sua resposta.

P23. Considere os protocolos GBN e SR. Suponha que o espaço de números de sequência seja de tamanho *k*. Qual será o maior tamanho de janela permissível que evitará que ocorram problemas como os da Figura 3.27 para cada um desses protocolos?

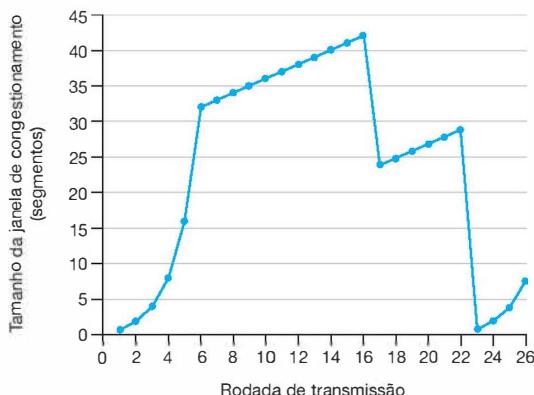
P24. Responda verdadeiro ou falso às seguintes perguntas e justifique resumidamente sua resposta:

a. Com o protocolo SR, é possível o remetente receber um ACK para um pacote que caia fora de sua janela corrente.

b. Com o GBN, é possível o remetente receber um ACK para um pacote que caia fora de sua janela corrente.

c. O protocolo bit alternante é o mesmo que o protocolo SR com janela do remetente e do destinatário de tamanho 1.

d. O protocolo bit alternante é o mesmo que o protocolo GBN com janela do remetente e do destinatário de tamanho 1.

P25. Dissemos que um aplicação pode escolher o UDP para um protocolo de transporte, pois oferece um controle de aplicações melhor (do que o TCP) de quais dados são enviados em um segmento e quando isso ocorre.

a. Por que uma aplicação possui mais controle de quais dados são enviados em um segmento?

b. Por que uma aplicação possui mais controle de quando o segmento é enviado?

P26. Considere a transferência de um arquivo enorme de *L* bytes do hospedeiro A para o hospedeiro B. Suponha um MSS de 536 bytes.

a. Qual é o máximo valor de *L* tal que não sejam esgotados os números de sequência TCP? Lembre-se de que o campo de número de sequência TCP tem 4 bytes.

b. Para o *L* que obtiver em (a), descubra quanto tempo demora para transmitir o arquivo. Admita que um total de 66 bytes de cabeçalho de transporte, de rede e de enlace de dados seja adicionado a cada segmento antes que o pacote resultante seja enviado por um enlace de 155 Mbits/s. Ignore controle de fluxo e controle de congestionamento de modo que A possa enviar os segmentos um atrás do outro e continuamente.

P27. Os hospedeiros A e B estão se comunicando por meio de uma conexão TCP, e o hospedeiro B já recebeu de A todos os bytes até o byte 126. Suponha que A envie, então, dois segmentos para B sucessivamente. O primeiro e o segundo segmentos contêm 80 e 40 bytes de dados. No primeiro segmento, o número de sequência é 127, o número de porta de partida é 302, e o número de porta de destino é 80. O hospedeiro B envia um reconhecimento ao receber um segmento do hospedeiro A.

a. No segundo segmento enviado do hospedeiro A para B, quais são o número de sequência, da porta de origem e da porta de destino?

b. Se o primeiro segmento chegar antes do segundo, no reconhecimento do primeiro segmento que chegar, qual é o número do reconhecimento, da porta de origem e da porta de destino?

c. Se o segundo segmento chegar antes do primeiro, no reconhecimento do primeiro segmento que chegar, qual é o número do reconhecimento?

d. Suponha que dois segmentos enviados por A cheguem em ordem a B. O primeiro reconhecimento é perdido e o segundo chega após o primeiro intervalo do esgotamento de temporização. Elabore um diagrama de temporização, mostrando esses segmentos, e todos os outros, e os reconhecimentos enviados. (Suponha que não haja qualquer perda de pacote adicional.) Para cada segmento de seu desenho, apresente o número de sequência e o número de bytes de dados; para cada reconhecimento adicionado por você, informe o número do reconhecimento.

P28. Os hospedeiros A e B estão diretamente conectados com um enlace de 100 Mbits/s. Existe uma conexão TCP entre os dois hospedeiros, e A está enviando a B um arquivo enorme por meio dessa conexão. O hospedeiro A pode enviar seus dados da aplicação para o *socket* TCP a uma taxa que chega a 120 Mbits/s, mas o hospedeiro B pode ler o buffer de recebimento TCP a uma taxa de 50 Mbits/s. Descreva o efeito do controle de fluxo do TCP.

P29. Os *cookies* SYN foram discutidos na Seção 3.5.6.

a. Por que é necessário que o servidor use um número de sequência especial no SYNACK?

b. Suponha que um atacante saiba que um hospedeiro-alvo utilize SYN *cookies*. O atacante consegue criar conexões semiabertas ou completamente abertas apenas enviando um pacote ACK para o alvo? Por quê?

c. Suponha que um atacante receba uma grande quantidade de números de sequência enviados pelo servidor. O atacante consegue fazer que o servidor crie muitas conexões totalmente abertas enviando ACKs com esses números de sequência? Por quê?

P30. Considere a rede mostrada no Cenário 2 na Seção 3.6.1. Suponha que os hospedeiros emissores A e B possuam valores de esgotamento de temporização fixos.

a. Analise o fato de que aumentar o tamanho do buffer finito do roteador pode possivelmente reduzir a vazão (λ_{out}).

b. Agora suponha que os hospedeiros ajustem dinamicamente seus valores de esgotamento de temporização (como o que o TCP faz) baseado no atraso no buffer no roteador. Aumentar o tamanho do buffer ajudaria a aumentar a vazão? Por quê?

P31. Suponha que os cinco valores de `SampleRTT` medidos (ver Seção 3.5.3) sejam 106 ms, 120 ms, 140 ms, 90 ms e 115 ms. Calcule o `EstimatedRTT` depois que forem obtidos cada um desses valores de `SampleRTT`, usando um valor de α = 0,125 e supondo que o valor de `EstimatedRTT` seja 100 ms imediatamente antes que a primeira dessas cinco amostras seja obtida. Calcule também o `DevRTT` após a obtenção de cada amostra, considerando um valor de β = 0,25 e que o valor de `DevRTT` seja 5 ms imediatamente antes que a primeira dessas cinco amostras seja obtida. Por fim, calcule o `TimeoutInterval` do TCP após a obtenção de cada uma dessas amostras.

P32. Considere o procedimento do TCP para estimar o RTT. Suponha que α = 0,1. Considere que $\texttt{SampleRTT}_1$ seja a amostra de RTT mais recente, $\texttt{SampleRTT}_2$ seja a próxima amostra mais recente, e assim por diante.

a. Para uma dada conexão TCP, suponha que quatro reconhecimentos foram devolvidos com as amostras RTT correspondentes `SampleRTT`$_4$, `SampleRTT`$_3$, `SampleRTT`$_2$ e `SampleRTT`$_1$. Expresse `EstimatedRTT` em termos das quatro amostras RTT.

b. Generalize sua fórmula para *n* amostras de RTTs.

c. Para a fórmula em (b), considere *n* tendendo ao infinito. Comente por que esse procedimento de média é denominado média móvel exponencial.

P33. Na Seção 3.5.3 discutimos estimativa de RTT para o TCP. Em sua opinião, por que o TCP evita medir o `SampleRTT` para segmentos retransmitidos?

P34. Qual é a relação entre a variável `SendBase` na Seção 3.5.4 e a variável `LastByteRcvd` na Seção 3.5.5?

P35. Qual é a relação entre a variável `LastByteRcvd` na Seção 3.5.5 e a variável y na Seção 3.5.4?

P36. Na Seção 3.5.4 vimos que o TCP espera até receber três ACKs duplicados antes de realizar uma retransmissão rápida. Em sua opinião, por que os projetistas do TCP preferiram não realizar uma retransmissão rápida após ser recebido o primeiro ACK duplicado para um segmento?

P37. Compare o GBN, SR e o TCP (sem ACK retardado). Admita que os valores do esgotamento de temporização para os três protocolos sejam longos o suficiente de tal modo que cinco segmentos de dados consecutivos e seus ACKs correspondentes possam ser recebidos (se não perdidos no canal) por um hospedeiro receptor (hospedeiro B) e por um hospedeiro emissor (hospedeiro A), respectivamente. Suponha que A envie cinco segmentos de dados para B, e que o segundo segmento (enviado de A) esteja perdido. No fim, todos os cinco segmentos de dados foram corretamente recebidos pelo hospedeiro B.

a. Quantos segmentos A enviou no total e quantos ACKs o hospedeiro B enviou no total? Quais são seus números de sequência? Responda essa questão para todos os três protocolos.

b. Se os valores do esgotamento de temporização para os três protocolos forem muito maiores do que 5 RTT, então qual protocolo envia com sucesso todos os cinco segmentos de dados em um menor intervalo de tempo?

P38. Em nossa descrição sobre o TCP na Figura 3.53, o valor do limiar, `ssthresh`, é definido como `ssthresh = cwnd/2` em diversos lugares e o valor `ssthresh` é referido como sendo definido para metade do tamanho da janela quando ocorreu um evento de perda. A taxa à qual o emissor está enviando quando ocorreu o evento de perda deve ser mais ou menos igual a segmentos `cwnd` por RTT? Explique sua resposta. Se for negativa, você pode sugerir uma maneira diferente pela qual `ssthresh` deva ser definido?

P39. Considere a Figura 3.46(b). Se λ'_{in} aumentar para mais do que $R/2$, λ_{out} poderá aumentar para mais do que $R/3$? Explique. Agora considere a Figura 3.46(c). Se λ'_{in} aumentar para mais do que $R/2$, λ_{out} poderá aumentar para mais de $R/4$ admitindo-se que um pacote será transmitido duas vezes, em média, do roteador para o destinatário? Explique.

P40. Considere a Figura 3.58. Admitindo-se que TCP Reno é o protocolo que experimenta o comportamento mostrado no gráfico, responda às seguintes perguntas. Em todos os casos você deverá apresentar uma justificativa resumida para sua resposta.

a. Quais os intervalos de tempo em que a partida lenta do TCP está em execução?

b. Quais os intervalos de tempo em que a prevenção de congestionamento do TCP está em execução?

c. Após a 16ª rodada de transmissão, a perda de segmento será detectada por três ACKs duplicados ou por um esgotamento de temporização?

d. Após a 22ª rodada de transmissão, a perda de segmento será detectada por três ACKs duplicados ou por um esgotamento de temporização?

e. Qual é o valor inicial de `ssthresh` na primeira rodada de transmissão?

FIGURA 3.58 TAMANHO DA JANELA TCP EM RELAÇÃO AO TEMPO

f. Qual é o valor inicial de `ssthresh` na 18ª rodada de transmissão?

g. Qual é o valor de `ssthresh` na 24ª rodada de transmissão?

h. Durante qual rodada de transmissão é enviado o 70º segmento?

i. Admitindo-se que uma perda de pacote será detectada após a 26ª rodada pelo recebimento de três ACKs duplicados, quais serão os valores do tamanho da janela de congestionamento e de `ssthresh`?

j. Suponha que o TCP Tahoe seja usado (em vez do TCP Reno) e que ACKs duplicados triplos sejam recebidos na 16ª rodada. Quais são o `ssthresh` e o tamanho da janela de congestionamento na 19ª rodada?

k. Suponha novamente que o TCP Tahoe seja usado, e que exista um evento de esgotamento de temporização na 22ª sessão. Quantos pacotes foram enviados da 17ª sessão até a 22ª, inclusive?

P41. Consulte a Figura 3.56, que ilustra a convergência do algoritmo AIMD do TCP. Suponha que, em vez de uma diminuição multiplicativa, o TCP reduza o tamanho da janela de uma quantidade constante. O AIMD resultante convergiria a um algoritmo de igual compartilhamento? Justifique sua resposta usando um diagrama semelhante ao da Figura 3.56.

P42. Na Seção 3.5.4 discutimos a duplicação do intervalo de temporização após um evento de esgotamento de temporização. Esse mecanismo é uma forma de controle de congestionamento. Por que o TCP precisa de um mecanismo de controle de congestionamento que utiliza janelas (como estudado na Seção 3.7) além desse mecanismo de duplicação do intervalo de esgotamento de temporização?

P43. O hospedeiro A está enviando um arquivo enorme ao hospedeiro B por uma conexão TCP. Nessa conexão nunca há perda de pacotes e os temporizadores nunca se esgotam. Seja R bits/s a taxa de transmissão do enlace que liga o hospedeiro A à Internet. Suponha que o processo em A consiga enviar dados para seu *socket* TCP a uma taxa de S bits/s, em que $S = 10 \cdot R$. Suponha ainda que o buffer de recepção do TCP seja grande o suficiente para conter o arquivo inteiro e que o buffer de envio possa conter apenas 1% do arquivo. O que impediria o hospedeiro A de passar dados continuamente para seu *socket* TCP à taxa de S bits/s: o controle de fluxo do TCP; o controle de congestionamento do TCP; ou alguma outra coisa? Elabore sua resposta.

P44. Considere enviar um arquivo grande de um computador a outro por meio de uma conexão TCP em que não haja perda.

a. Suponha que o TCP utilize AIMD para seu controle de congestionamento sem partida lenta. Admitindo que `cwnd` aumenta 1 MSS sempre que um lote de ACK é recebido e os tempos da viagem de ida e volta

constantes, quanto tempo leva para `cwnd` aumentar de 6 MSS para 12 MSS? (admitindo nenhum evento de perda)?

b. Qual é a vazão média (em termos de MSS e RTT) para essa conexão sendo o tempo = 6 RTT?

P45. Relembre a descrição macroscópica da vazão do TCP. No período de tempo transcorrido para a taxa da conexão variar de $W/(2 \cdot RTT)$ a W/RTT, apenas um pacote é perdido (bem ao final do período).

a. Mostre que a taxa de perda (fração de pacotes perdidos) é igual a

$$L = \text{taxa de perda} = \frac{1}{\frac{3}{8}W^2 + \frac{3}{4}W}$$

b. Use o resultado anterior para mostrar que, se uma conexão tiver taxa de perda L, sua largura de banda média é dada aproximadamente por:

$$\approx \frac{1{,}22 \cdot MSS}{RTT\sqrt{L}}$$

P46. Considere que somente uma única conexão TCP (Reno) utiliza um enlace de 10 Mbits/s que não armazena nenhum dado. Suponha que esse enlace seja o único congestionado entre os hospedeiros emissor e receptor. Admita que o emissor TCP tenha um arquivo enorme para enviar ao receptor e o buffer de recebimento do receptor é muito maior do que a janela de congestionamento. Também fazemos as seguintes suposições: o tamanho de cada segmento TCP é 1.500 bytes; o atraso de propagação bidirecional dessa conexão é 150 ms; e essa conexão TCP está sempre na fase de prevenção de congestionamento, ou seja, ignore a partida lenta.

a. Qual é o tamanho máximo da janela (em segmentos) que a conexão TCP pode atingir?

b. Qual é o tamanho médio da janela (em segmentos) e a vazão média (em bits/s) dessa conexão TCP?

c. Quanto tempo essa conexão TCP leva para alcançar sua janela máxima novamente após se recuperar da perda de um pacote?

P47. Considere o cenário descrito na questão anterior. Suponha que o enlace de 10 Mbits/s possa armazenar um número finito de segmentos. Demonstre que para o enlace sempre enviar dados, teríamos que escolher um tamanho de buffer que é, pelo menos, o produto da velocidade do enlace C e o atraso de propagação bidirecional entre o emissor e o receptor.

P48. Repita a Questão 46, mas substituindo o enlace de 10 Mbits/s por um de 10 Gbits/s. Observe que em sua resposta ao item (c), verá que o tamanho da janela de congestionamento leva muito tempo para atingir seu tamanho máximo após se recuperar de uma perda de pacote. Elabore uma solução para resolver o problema.

P49. Suponha que T (medido por RTT) seja o intervalo de tempo que uma conexão TCP leva para aumentar seu tamanho de janela de congestionamento de $W/2$ para W, sendo W o tamanho máximo da janela de congestionamento. Demonstre que T é uma função da vazão média do TCP.

P50. Considere um algoritmo AIMD do TCP simplificado, sendo o tamanho da janela de congestionamento medido em número de segmentos e não em bytes. No aumento aditivo, o tamanho da janela de congestionamento aumenta por um segmento em cada RTT. Na diminuição multiplicativa, o tamanho da janela de congestionamento diminui para metade (se o resultado não for um número inteiro, arredondar para o número inteiro mais próximo). Suponha que duas conexões TCP, C_1 e C_2 compartilhem um único enlace congestionado com 30 segmentos por segundo de velocidade. Admita que C_1 e C_2 estejam na fase de prevenção de congestionamento. O RTT da conexão C_1 é de 100 ms e o da conexão C_2 é de 200 ms. Suponha que quando a taxa de dados no enlace excede a velocidade do enlace, todas as conexões TCP sofrem perda de segmento de dados.

a. Se C_1 e C_2 no tempo t_0 possui uma janela de congestionamento de 10 segmentos, quais são seus tamanhos de janela de congestionamento após 1.000 ms?

b. No final das contas, essas duas conexões obterão a mesma porção da largura de banda do enlace congestionado? Explique.

P51. Considere a rede descrita na questão anterior. Agora suponha que as duas conexões TCP, C_1 e C_2, possuam o mesmo RTT de 100 ms e que, no tempo t_0, o tamanho da janela de congestionamento de C_1 seja 15 segmentos, e que o de C_2 seja 10 segmentos.

a. Quais são os tamanhos de suas janelas de congestionamento após 2.200 ms?

b. No final das contas, essas duas conexões obterão a mesma porção da largura de banda do enlace congestionado?

c. Dizemos que duas conexões são sincronizadas se ambas atingirem o tamanho máximo e mínimo de janela ao mesmo tempo. No final das contas, essas duas conexões serão sincronizadas? Se sim, quais os tamanhos máximos de janela?

d. Essa sincronização ajudará a melhorar a utilização do enlace compartilhado? Por quê? Elabore alguma ideia para romper essa sincronização.

P52. Considere uma modificação ao algoritmo de controle de congestionamento do TCP. Em vez do aumento aditivo, podemos utilizar o aumento multiplicativo. Um emissor TCP aumenta seu tamanho de janela por uma constante positiva pequena a ($0 < a < 1$) ao receber um ACK válido. Encontre a relação funcional entre a taxa de perda L e a janela máxima de congestionamento W. Para esse TCP modificado, demonstre, independentemente da vazão média TCP, que uma conexão TCP sempre gasta a mesma quantidade de tempo para aumentar seu tamanho da janela de congestionamento de $W/2$ para W.

P53. Quando discutimos TCPs futuros na Seção 3.7, observamos que, para conseguir uma vazão de 10 Gbits/s, o TCP apenas poderia tolerar uma probabilidade de perda de segmentos de $2 \cdot 10^{-10}$ (ou, equivalentemente, uma perda para cada 5 milhões de segmentos). Mostre a derivação dos valores para $2 \cdot 10^{-10}$ (ou 1: 5.000.000) a partir dos valores de RTT e do MSS dados na Seção 3.7. Se o TCP precisasse suportar uma conexão de 100 Gbits/s, qual seria a perda tolerável?

P54. Quando discutimos controle de congestionamento em TCP na Seção 3.7, admitimos implicitamente que o remetente TCP sempre tinha dados para enviar. Agora considere o caso em que o remetente TCP envia uma grande quantidade de dados e então fica ocioso em t_1 (já que não há mais dados a enviar). O TCP permanecerá ocioso por um período de tempo relativamente longo e então irá querer enviar mais dados em t_2. Quais são as vantagens e desvantagens de o TCP utilizar os valores `cwnd` e `ssthresh` de t_1 quando começar a enviar dados em t_2? Que alternativa você recomendaria? Por quê?

P55. Neste problema, verificamos se o UDP ou o TCP apresentam um grau de autenticação do ponto de chegada.

a. Considere um servidor que receba uma solicitação dentro de um pacote UDP e responda a essa solicitação dentro de um pacote UDP (por exemplo, como feito por um servidor DNS). Se um cliente com endereço IP X o engana com o endereço Y, para onde o servidor enviará sua resposta?

b. Suponha que um servidor receba um SYN de endereço IP de origem Y, e depois de responder com um SYNACK, recebe um ACK com o endereço IP de origem Y com o número de reconhecimento correto. Admitindo que o servidor escolha um número de sequência aleatório e que não haja um "*man-in-the-middle*", o servidor pode ter certeza de que o cliente realmente está em Y (e não em outro endereço X que está se passando por Y)?

P56. Neste problema, consideramos o atraso apresentado pela fase de partida lenta do TCP. Considere um cliente e um servidor da Web diretamente conectados por um enlace de taxa R. Suponha que o cliente queira restaurar um objeto cujo tamanho seja exatamente igual a 15 S, sendo S o tamanho máximo do segmento (MSS). Considere RTT o tempo de viagem de ida e volta entre o cliente e o servidor (admitindo que seja constante). Ignorando os cabeçalhos do protocolo, determine o tempo para restaurar o objeto (incluindo o estabelecimento da conexão TCP) quando

a. $4\ S/R > S/R + RTT > 2S/R$

b. $S/R + RTT > 4\ S/R$

c. $S/R > RTT$.

TAREFA DE PROGRAMAÇÃO

Implementando um protocolo de transporte confiável

Nesta tarefa de programação de laboratório, você escreverá o código para a camada de transporte do remetente e do destinatário no caso da implementação de um protocolo simples de transferência confiável de dados. Há duas versões deste laboratório: a do protocolo de bit alternante e a do GBN. Essa tarefa será muito divertida, já que a sua realização não será muito diferente da que seria exigida em uma situação real.

Como você provavelmente não tem máquinas autônomas (com um sistema operacional que possa modificar), seu código terá de rodar em um ambiente de hardware/software simulado. Contudo, a interface de programação fornecida a suas rotinas — isto é, o código que chamaria suas entidades de cima e de baixo — é muito próxima ao que é feito em um ambiente UNIX real. (Na verdade, as interfaces do software descritas nesta tarefa de programação são muito mais realistas do que os remetentes e destinatários de laço infinito descritos em muitos livros.) A parada e o acionamento dos temporizadores também são simulados, e as interrupções do temporizador farão que sua rotina de tratamento de temporização seja ativada.

A tarefa completa de laboratório, assim como o código necessário para compilar seu próprio código, estão disponíveis no site: http://sv.pearson.com.br.

WIRESHARK LAB: EXPLORANDO O TCP

Neste laboratório, você usará seu navegador para acessar um arquivo de um servidor Web. Como nos anteriores, você usará Wireshark para capturar os pacotes que estão chegando ao seu computador. Mas, diferentemente daqueles laboratórios, *também* poderá baixar, do mesmo servidor Web do qual baixou o arquivo, um relatório (*trace*) de pacotes que pode ser lido pelo Wireshark. Nesse relatório do servidor, você encontrará os pacotes que foram gerados pelo seu próprio acesso ao servidor Web. Você analisará os diagramas dos lados do cliente e do servidor de modo a explorar aspectos do TCP. Particularmente, fará uma avaliação do desempenho da conexão TCP entre seu computador e o servidor Web. Você analisará o comportamento da janela TCP e deduzirá comportamentos de perda de pacotes, de retransmissão, de controle de fluxo e de controle de congestionamento e do tempo de ida e volta estimado.

Como acontece com todos os Wireshark Labs, a descrição completa deste pode ser encontrada, em inglês, no site http://sv.pearson.com.br.

WIRESHARK LAB: EXPLORANDO O UDP

Neste pequeno laboratório, você realizará uma captura de pacote e uma análise de sua aplicação favorita que utiliza o UDP (por exemplo, o DNS ou uma aplicação multimídia, como o Skype). Como aprendemos na Seção 3.3, o UDP é um protocolo de transporte simples. Neste laboratório, você examinará os campos do cabeçalho no segmento UDP, assim como o cálculo da soma de verificação.

Como acontece com todos os Wireshark Labs, a descrição completa deste pode ser encontrada, em inglês, no site http://sv.pearson.com.br.

ENTREVISTA

Van Jacobson

Van Jacobson é Research Fellow no PARC. Antes disso, foi cofundador e cientista chefe da Packet Design. Antes ainda, foi cientista chefe na Cisco. Antes de entrar para a Cisco, chefiou o Network Research Group no Lawrence Berkeley National Laboratory e lecionou na Universidade da Califórnia em Berkeley e Stanford. Van recebeu o prêmio ACM SIGCOMM em 2001 pela destacada contribuição de toda uma vida para o campo de redes de comunicação e o prêmio IEEE Kobayashi em 2002 por "contribuir para o conhecimento do congestionamento de redes e por desenvolver mecanismos de controle de congestionamento de rede que permitiram a escalada bem-sucedida da Internet". Em 2004, foi eleito para a Academia Nacional de Engenharia dos Estados Unidos.

Por favor, descreva um ou dois dos projetos mais interessantes em que você já trabalhou durante a sua carreira. Quais foram os maiores desafios?

A escola nos ensina muitas maneiras de achar respostas. Em cada problema interessante em que trabalhei, o desafio tem sido achar a pergunta certa. Quando Mike Karels e eu começamos a examinar o congestionamento do TCP, gastamos meses encarando o protocolo e os *traces* de pacotes, perguntando "por que ele está falhando"? Um dia, no escritório de Mike, um de nós disse: "O motivo pelo qual não consigo descobrir por que ele falha é porque não entendo como ele chegou a funcionar, para começar". Aquela foi a pergunta certa e nos forçou a compreender a "temporização dos reconhecimentos (ACK)" que faz o TCP funcionar. Depois disso, o resto foi fácil.

De modo geral, qual é o futuro que você imagina para as redes e a Internet?

Para a maioria das pessoas, a Web é a Internet. O pessoal que trabalha com redes sorri educadamente, pois sabemos que a Web é uma aplicação rodando sobre a Internet, mas, e se eles estiverem certos? A Internet trata de permitir conversações entre pares de hospedeiros. A Web trata da produção e consumo de informações distribuídas. "Propagação de informações" é uma visão muito geral da comunicação, da qual a "conversa em pares" é um minúsculo subconjunto. Precisamos pensar além do que vemos. As redes de hoje lidam com a mídia de *broadcast* (rádios, PONs etc.) fingindo que ela é um fio de ponto a ponto. Isso é tremendamente ineficaz. Terabits de dados por segundo estão sendo trocados pelo mundo inteiro por meio de *pendrives* ou smartphones, mas não sabemos como tratar isso como "rede". Os ISPs estão ocupados montando *caches* e CDNs para distribuir vídeo e áudio de modo a facilitar a expansão. O *caching* é uma parte necessária da solução, mas não há uma parte das redes de hoje — desde Informações, Enfileiramento ou Teoria de Tráfego até as especificações de protocolos da Internet — que nos diga como projetá-lo e distribuí-lo. Acho e espero que, nos próximos anos, as redes evoluam para abranger a visão muito maior de comunicação, que é a base da Web.

Que pessoas o inspiraram profissionalmente?

Quando eu cursava a faculdade, Richard Feynman nos visitou e deu um seminário acadêmico. Ele falou sobre uma parte da teoria quântica que eu estava lutando para entender durante todo o semestre, e sua explicação foi tão simples e lúcida que aquilo que parecia um lixo sem sentido para mim tornou-se óbvio e inevitável. Essa capacidade de ver e transmitir a simplicidade que está por trás do nosso mundo complexo me parece uma dádiva rara e maravilhosa.

Quais são suas recomendações para estudantes que desejam seguir carreira em computação e tecnologia da informação?

Este é um campo maravilhoso — computadores e redes provavelmente tiveram mais impacto sobre a sociedade do que qualquer invenção desde a imprensa. Redes conectam coisas, e seu estudo o ajuda a fazer conexões intelectuais: a busca de alimento das formigas e as danças das abelhas demonstram o projeto de proto-

colo melhor do que RFCs, engarrafamentos de trânsito ou pessoas saindo de um estádio lotado são a essência do congestionamento, e motoristas procurando o melhor caminho de volta para casa após uma tempestade que alagou a cidade constituem o núcleo do roteamento dinâmico. Se você estiver interessado em obter muito material e quiser causar impacto, é difícil imaginar um campo melhor do que este.

CAPÍTULO

1 2 3 4 5 6 7 8 9

A CAMADA DE **REDE**

Vimos no capítulo anterior que a camada de transporte oferece várias formas de comunicação processo a processo com base no serviço de comunicação entre hospedeiros da camada de rede. Vimos também que a camada de transporte faz isso sem saber como a camada de rede implementa esse serviço. Portanto, é bem possível que agora você esteja imaginando o que está por baixo do serviço de comunicação hospedeiro a hospedeiro; o que o faz funcionar.

Neste capítulo estudaremos exatamente como a camada de rede executa o serviço de comunicação hospedeiro a hospedeiro. Veremos que há um pedaço da camada de rede em cada hospedeiro e roteador na rede, o que não acontece com as camadas de transporte e de aplicação. Por causa disso, os protocolos de camada de rede estão entre os mais desafiadores (e, portanto, os mais interessantes!) da pilha de protocolos.

A camada de rede é, também, uma das mais complexas da pilha de protocolos e, assim, temos um longo caminho a percorrer. Iniciaremos nosso estudo com uma visão geral da camada de rede e dos serviços que ela pode prover. Em seguida, examinaremos novamente as duas abordagens da estruturação da entrega de pacotes de camada de rede — o modelo de datagramas e o modelo de circuitos virtuais — que vimos pela primeira vez no Capítulo 1 e veremos o papel fundamental que o endereçamento desempenha na entrega de um pacote a seu hospedeiro de destino.

Faremos, neste capítulo, uma distinção importante entre as funções de **repasse** e **roteamento** da camada de rede. Repasse envolve a transferência de um pacote de um enlace de entrada para um enlace de saída dentro de um *único* roteador. Roteamento envolve *todos* os roteadores de uma rede, cujas interações coletivas por meio do protocolo de roteamento determinam os caminhos que os pacotes percorrem em suas viagens do nó de origem ao de destino. Essa será uma distinção importante para manter em mente ao ler este capítulo.

Para aprofundar nosso conhecimento do repasse de pacotes, examinaremos o "interior" de um roteador — a organização e a arquitetura de seu hardware. Então, estudaremos o repasse de pacotes na Internet, junto com o famoso Protocolo da Internet (IP). Investigaremos o endereçamento na camada de rede e o formato de datagrama IPv4. Em seguida, estudaremos a tradução de endereço de rede (*Network Address Translation* — NAT), a fragmentação de datagrama, o Protocolo de Mensagem de Controle da Internet (*Internet Control Message Protocol* — ICMP) e IPv6.

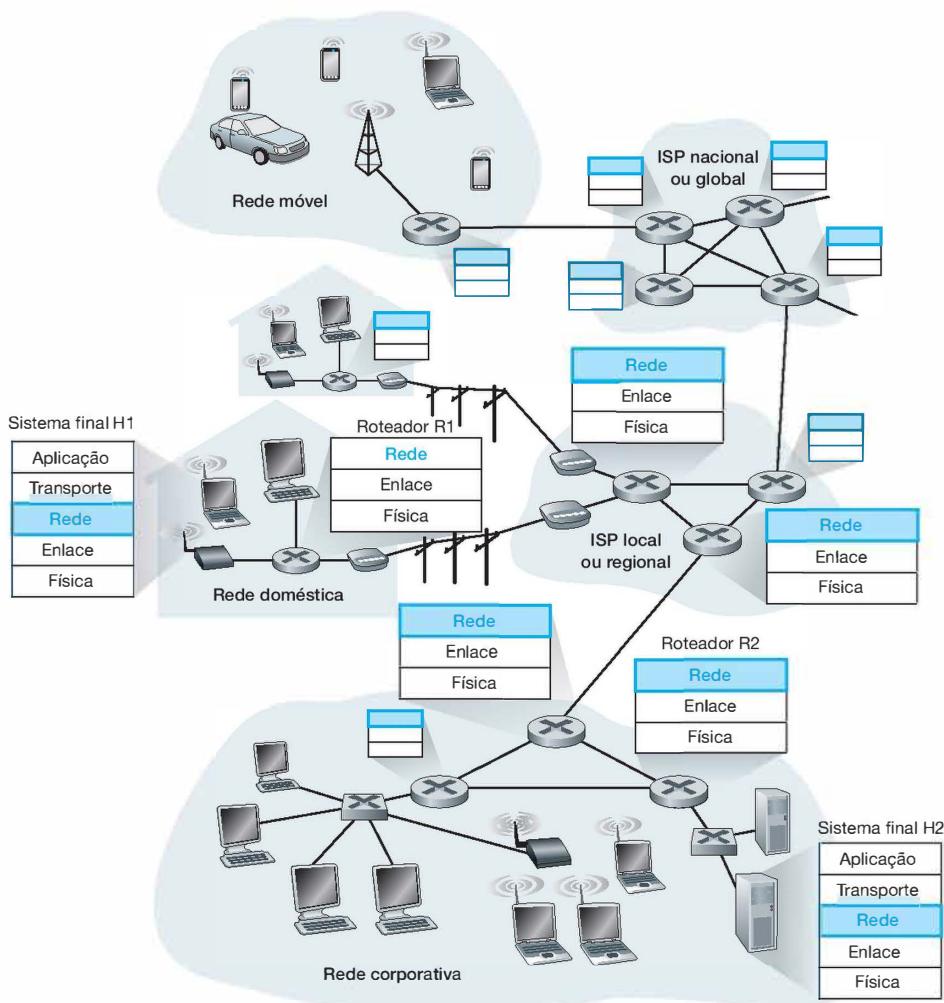
Então voltaremos nossa atenção à função de roteamento da camada de rede. Veremos que o trabalho de um algoritmo de roteamento é determinar bons caminhos (ou rotas) entre remetentes e destinatários. Em primeiro lugar, estudaremos a teoria de algoritmos de roteamento concentrando nossa atenção nas duas classes mais pre-

dominantes: algoritmos de estado de enlace e de vetor de distâncias. Visto que a complexidade dos algoritmos de roteamento cresce consideravelmente com o aumento do número de roteadores na rede, também será interessante abordar o roteamento hierárquico. Em seguida veremos como a teoria é posta em prática quando falarmos sobre os protocolos de roteamento de intrassistemas autônomos da Internet (RIP, OSPF e IS-IS) e seu protocolo de roteamento de intersistemas autônomos, o BGP. Encerraremos este capítulo com uma discussão sobre roteamento por difusão (*broadcast*) e para um grupo (*multicast*).*

Em resumo, este capítulo tem três partes importantes. A primeira, seções 4.1 e 4.2, aborda funções e serviços de camada de rede. A segunda, seções 4.3 e 4.4, examina o repasse. Por fim, a terceira parte, seções 4.5 a 4.7, estuda o roteamento.

4.1 INTRODUÇÃO

A Figura 4.1 mostra uma rede simples com dois hospedeiros, H1 e H2, e diversos roteadores no caminho entre H1 e H2. Suponha que H1 esteja enviando informações a H2 e considere o papel da camada de rede nesses hospedeiros e nos roteadores intervenientes. A camada de rede em H1 pegará segmentos da camada de transporte em H1, encapsulará cada segmento em um datagrama (isto é, em um pacote de camada de rede) e então dará início à jornada dos datagramas até seu destino, isto é, ela os enviará para seu roteador vizinho, R1. No hospedeiro receptor (H2), a camada de rede receberá os datagramas de seu roteador vizinho R2, extrairá os segmentos de camada de transporte e os entregará à camada de transporte em H2. O papel primordial dos roteadores é repassar datagramas de enlaces de entrada para enlaces de saída. Note que os da Figura 4.1 são mostrados com a pilha de protocolos truncada, isto é, sem as camadas superiores acima da camada de rede, porque (exceto para finalidades de controle) roteadores não rodam protocolos de camada de transporte e de aplicação como os que examinamos nos Capítulos 2 e 3.

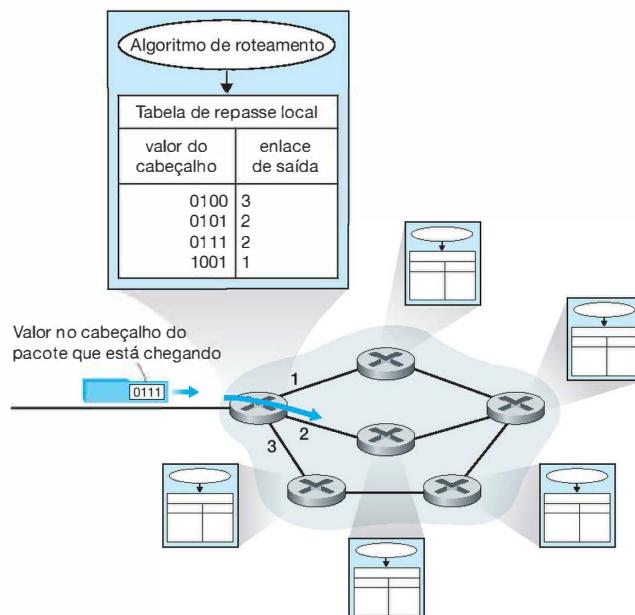
4.1.1 Repasse e roteamento

Assim, o papel da camada de rede é aparentemente simples — transportar pacotes de um hospedeiro remetente a um hospedeiro destinatário. Para fazê-lo, duas importantes funções da camada de rede podem ser identificadas:

- *Repasse.* Quando um pacote chega ao enlace de entrada de um roteador, este deve conduzi-lo até o enlace de saída apropriado. Por exemplo, um pacote proveniente do hospedeiro H1 que chega ao roteador R1 deve ser repassado ao roteador seguinte por um caminho até H2. Na Seção 4.3 examinaremos o interior de um roteador e investigaremos como um pacote é realmente repassado de um enlace de entrada de um roteador até um enlace de saída.
- *Roteamento.* A camada de rede deve determinar a rota ou o caminho tomado pelos pacotes ao fluírem de um remetente a um destinatário. Os algoritmos que calculam esses caminhos são denominados **algoritmos de roteamento**. Um algoritmo de roteamento determinaria, por exemplo, o caminho pelo qual os pacotes fluiriam de H1 para H2.

Os termos *repasse* e *roteamento* são usados indistintamente por autores que estudam a camada de rede. Neste livro, usaremos tais termos com maior exatidão. *Repasse* refere-se à ação local realizada por um roteador para transferir um pacote da interface de um enlace de entrada para a interface de enlace de saída apropriada. *Roteamento* refere-se ao processo de âmbito geral da rede que determina os caminhos fim a fim que os pacotes percorrem desde a origem até o destino. Para usar uma viagem como analogia, voltemos àquele nosso viajante da

* Não há uma tradução única e conclusiva sobre estes termos. Muitos textos não os traduzem. Aqui, procuramos usar a tradução que mais transmita seu sentido. Assim, usamos endereço/enlace/roteamento/transmissão/rede "por difusão" (*broadcast*), "para um grupo" (*multicast*), "individual" (*unicast*) e "para um membro do grupo" (*anycast*). (N. do T.)

FIGURA 4.1 A CAMADA DE REDE

Seção 1.3.1, que vai da Pensilvânia à Flórida. Durante a viagem, nosso motorista passa por muitos cruzamentos de rodovias em sua rota. Podemos imaginar o repasse como o processo de passar por um único cruzamento: um carro chega ao cruzamento vindo de uma rodovia e determina qual rodovia ele deve pegar para sair do cruzamento. Podemos imaginar o roteamento como o processo de planejamento da viagem da Pensilvânia até a Flórida: antes de partir, o motorista consultou um mapa e escolheu um dos muitos caminhos possíveis. Cada um deles consiste em uma série de trechos de rodovias conectados por cruzamentos.

Cada roteador tem uma **tabela de repasse**. Um roteador repassa um pacote examinando o valor de um campo no cabeçalho do pacote que está chegando e então utiliza esse valor para indexar sua tabela de repasse. O resultado da tabela de repasse indica para qual das interfaces de enlace do roteador o pacote deve ser repassado. Dependendo do protocolo de camada de rede, o valor no cabeçalho do pacote pode ser o endereço de destino do pacote ou uma

indicação da conexão à qual ele pertence. A Figura 4.2 dá um exemplo. Nessa figura, um pacote cujo valor no campo de cabeçalho é 0111 chega a um roteador. Este o indexa em sua tabela de repasse, determina que a interface de enlace de saída para o pacote é a interface 2 e, então, o repassa internamente à interface 2. Na Seção 4.3 examinaremos o interior de um roteador e estudaremos a função de repasse muito mais detalhadamente.

Agora você deve estar imaginando como são configuradas as tabelas de repasse nos roteadores. Essa é uma questão crucial, que expõe a importante interação entre roteamento e repasse. Como ilustrado na Figura 4.2, o algoritmo de roteamento determina os valores que são inseridos nas tabelas de repasse dos roteadores. Esse algoritmo pode ser centralizado (por exemplo, com um algoritmo que roda em um local central e descarrega informações de roteamento a cada um dos roteadores) ou descentralizado (isto é, com um pedaço do algoritmo de roteamento distribuído funcionando em cada roteador). Em qualquer dos casos, um roteador recebe mensagens de protocolo de roteamento que são utilizadas para configurar sua tabela de repasse. As finalidades distintas e diferentes das funções de repasse e roteamento podem ser mais bem esclarecidas considerando o caso hipotético (e não realista, mas tecnicamente viável) de uma rede na qual todas as tabelas de repasse são configuradas diretamente por operadores de rede humanos, fisicamente presentes nos roteadores. Nesse caso, não seria preciso *nenhum* protocolo de roteamento! É claro que os operadores humanos precisariam interagir uns com os outros para garantir que as tabelas fossem configuradas de tal modo que os pacotes chegassem a seus destinos pretendidos. Também é provável que uma configuração humana seria mais propensa a erro e muito mais lenta do que um protocolo de roteamento para reagir a mudanças na topologia da rede. Portanto, sorte nossa que todas as redes têm uma função de repasse *e também* uma função de roteamento!

Enquanto estamos no tópico da terminologia, é interessante mencionar dois outros termos que também são utilizados indistintamente, mas que usaremos com maior cuidado. Reservaremos o termo *comutador de pacotes* para designar um dispositivo geral de comutação de pacotes que transfere um pacote de interface de enlace de entrada para interface de enlace de saída conforme o valor que está em um campo no cabeçalho do pacote. Alguns comutadores de pacotes, denominados **comutadores de camada de enlace** (que vere-

FIGURA 4.2 ALGORITMOS DE ROTEAMENTO DETERMINAM VALORES EM TABELAS DE REPASSE

Algoritmo de roteamento

Tabela de repasse local

valor do cabeçalho	enlace de saída
0100	3
0101	2
0111	2
1001	1

Valor no cabeçalho do pacote que está chegando

0111

mos no Capítulo 5), baseiam a decisão de repasse no valor que está no campo da camada de enlace. Outros, denominados **roteadores**, baseiam sua decisão de repasse no valor que está no campo de camada de rede. Os roteadores, portanto, são dispositivos da camada de rede (camada 3), mas também devem utilizar protocolos da camada 2, pois os dispositivos da camada 3 exigem os serviços da camada 2 para implementar sua funcionalidade (camada 3). (Para dar real valor a essa importante distinção, seria interessante você ler novamente a Seção 1.5.2, em que discutimos datagramas de camada de rede e quadros de camada de enlace e as relações entre eles.) Visto que o foco deste capítulo é a camada de rede, usaremos o termo *roteador* no lugar de *comutador de pacotes*. Usaremos o termo *roteador* até mesmo quando falarmos sobre comutadores de pacotes em redes de circuitos virtuais (que discutiremos em breve).

Estabelecimento de conexão

Acabamos de dizer que a camada de rede tem duas funções importantes, repasse e roteamento. Mas logo veremos que em algumas redes de computadores há uma terceira função importante, a saber, o **estabelecimento de conexão**. Lembre-se de que, quando estudamos o TCP, verificamos que é necessária uma apresentação de três vias antes de os dados realmente poderem fluir do remetente ao destinatário. Isso permite que o remetente e o destinatário estabeleçam a informação de estado necessária (por exemplo, número de sequência e tamanho inicial da janela de controle de fluxo). De modo semelhante, algumas arquiteturas de camada de rede — por exemplo, ATM, *frame-relay* e MPLS (que estudaremos na Seção 5.8) — exigem que roteadores ao longo do caminho escolhido desde a origem até o destino troquem mensagens entre si com a finalidade de estabelecer estado antes que pacotes de dados de camada de rede dentro de uma dada conexão origem-destino possam começar a fluir. Na camada de rede, esse processo é denominado *estabelecimento de conexão*. Examinaremos estabelecimento de conexão na Seção 4.2.

4.1.2 Modelos de serviço de rede

Antes de examinar a camada de rede, vamos tomar uma perspectiva mais ampla e considerar os diferentes tipos de serviço que poderiam ser oferecidos por ela. Quando a camada de transporte em um hospedeiro remetente transmite um pacote para dentro da rede (isto é, passa o pacote para a camada de rede do hospedeiro remetente), ela pode contar com a camada de rede para entregar o pacote no destino? Quando são enviados vários pacotes, eles serão entregues à camada de transporte no hospedeiro destinatário na ordem em que foram enviados? A quantidade de tempo decorrido entre duas transmissões de pacotes sequenciais será a mesma quantidade de tempo decorrido entre suas recepções? A rede fornecerá algum tipo de informação sobre congestionamento na rede? Qual é o modelo (propriedades) abstrato do canal que conecta a camada de transporte nos hospedeiros remetente e destinatário? As respostas a essas e a outras perguntas são determinadas pelo modelo de serviço oferecido pela camada de rede. O **modelo de serviço de rede** define as características do transporte de dados fim a fim entre uma borda da rede e a outra, isto é, entre sistemas finais remetente e destinatário.

Vamos considerar agora alguns serviços possíveis que a camada de rede poderia prover. No hospedeiro remetente, quando a camada de transporte passa um pacote para a camada de rede, alguns serviços específicos que poderiam ser oferecidos são:

- *Entrega garantida*. Esse serviço assegura que o pacote mais cedo ou mais tarde chegará a seu destino.
- *Entrega garantida com atraso limitado*. Não somente assegura a entrega de um pacote, mas também a entrega com um atraso hospedeiro a hospedeiro limitado e especificado (por exemplo, dentro de 100 ms).

Além disso, há outros serviços que podem ser providos a um *fluxo de pacotes* entre uma origem e um destino determinados, como os seguintes:

- *Entrega de pacotes na ordem*. Garante que pacotes chegarão ao destino na ordem em que foram enviados.
- *Largura de banda mínima garantida*. Esse serviço de camada de rede emula o comportamento de um enlace de transmissão com uma taxa de bits especificada (por exemplo, 1 bit/s) entre hospedeiros reme-

tentes e destinatários. Contanto que o hospedeiro remetente transmita bits (como parte de pacotes) a uma taxa abaixo da taxa de bits especificada, nenhum pacote será perdido e cada um chegará dentro de um atraso hospedeiro a hospedeiro previamente especificado (por exemplo, dentro de 40 ms).

- *Jitter máximo garantido*. Assegura que a quantidade de tempo entre a transmissão de dois pacotes sucessivos no remetente seja igual à quantidade de tempo entre o recebimento dos dois pacotes no destino (ou que esse espaçamento não mude mais do que algum valor especificado).
- *Serviços de segurança*. Utilizando uma chave de sessão secreta conhecida somente por um hospedeiro de origem e de destino, a camada de rede no computador de origem pode codificar a carga útil de todos os datagramas que estão sendo enviados ao computador de destino. A camada de rede no computador de destino, então, seria responsável por decodificar as cargas úteis. Com esse serviço, o sigilo seria fornecido para todos os segmentos da camada de transporte (TCP e UDP) entre os computadores de origem e de destino. Além do sigilo, a camada de rede poderia prover integridade dos dados e serviços de autenticação na origem.

Essa é uma lista apenas parcial de serviços que uma camada de rede poderia prover — há incontáveis variações possíveis.

A camada de rede da Internet fornece um único modelo de serviço, conhecido como **serviço de melhor esforço**. Consultando a Tabela 4.1, pode parecer que *serviço de melhor esforço* seja um eufemismo para *absolutamente nenhum serviço*. Com o serviço de melhor esforço, não há garantia de que a temporização entre pacotes seja preservada, não há garantia de que os pacotes sejam recebidos na ordem em que foram enviados e não há garantia da entrega final dos pacotes transmitidos. Dada essa definição, uma rede que não entregasse *nenhum* pacote ao destinatário satisfaria a definição de serviço de entrega de melhor esforço. Contudo, como discutiremos adiante, há razões sólidas para esse modelo minimalista de serviço de camada de rede.

Outras arquiteturas de rede definiram e puseram em prática modelos de serviço que vão além do serviço de melhor esforço da Internet. Por exemplo, a arquitetura de rede ATM [MFA Forum 2012; Black, 1995] habilita vários modelos de serviço, o que significa que, dentro da mesma rede, podem ser oferecidas conexões diferentes com classes de serviço diferentes. Discutir o modo como uma rede ATM oferece esses serviços vai muito além do escopo deste livro; nossa meta aqui é apenas salientar que existem alternativas ao modelo de melhor esforço da Internet. Dois dos modelos mais importantes de serviço ATM são os serviços de taxa constante e de taxa disponível.

- ***Serviço de rede de taxa constante de bits (Constant Bit Rate — CBR).*** Foi o primeiro modelo de serviço ATM a ser padronizado, refletindo o interesse imediato das empresas de telefonia por esse serviço e a adequação do serviço CBR para transmitir tráfego de áudio e vídeo de taxa constante de bits. O objetivo é conceitualmente simples: prover um fluxo de pacotes (conhecidos como células na terminologia ATM) com uma tubulação virtual cujas propriedades são iguais às de um hipotético enlace de transmissão dedicado de largura de banda fixa entre os hospedeiros remetente e destinatário. Com serviço CBR, um fluxo de células ATM é carregado através da rede de modo tal que garanta que o atraso fim a fim de uma

TABELA 4.1 MODELOS DE SERVIÇO DAS REDES INTERNET, ATM CBR E ATM ABR

Arquitetura da rede	Modelo de serviço	Garantia de largura de banda	Garantia contra perda	Ordenação	Temporização	Indicação de congestionamento
Internet	Melhor esforço	Nenhuma	Nenhuma	Qualquer ordem possível	Não mantida	Nenhuma
ATM	CBR	Taxa constante garantida	Sim	Na ordem	Mantida	Não haverá congestionamento
ATM	ABR	Mínima garantida	Nenhuma	Na ordem	Não mantida	Indicação de congestionamento

célula, a variabilidade do atraso (isto é, o *jitter*) e a fração de células perdidas ou entregues atrasadas sejam menores do que valores especificados. Tais valores são acertados entre o hospedeiro remetente e a rede ATM quando a conexão CBR é estabelecida pela primeira vez.

- ***Serviço de rede de taxa de bits disponível** (Available Bit Rate* — ABR). Como o serviço oferecido pela Internet é "de melhor esforço", o ATM ABR pode ser caracterizado como um "serviço de melhor esforço ligeiramente melhorado". Como acontece com o modelo de serviço da Internet, também com o serviço ABR pode haver perda de células. Mas, ao contrário da Internet, as células não podem ser reordenadas (embora possam ser perdidas) e é garantida uma taxa mínima de transmissão de células (*minimum cell transmission rate* — MCR) para uma conexão que está usando o serviço ABR. Se, contudo, a rede dispuser de suficientes recursos livres em dado momento, um remetente também poderá enviar células com sucesso a uma taxa mais alta do que a MCR. Além disso, como vimos na Seção 3.6, o serviço ATM ABR pode prover realimentação ao remetente (em termos de um bit de notificação de congestionamento ou de uma taxa de envio explícita), que controla o modo como o remetente ajusta sua taxa entre a MCR e uma taxa de pico admissível.

4.2 REDES DE CIRCUITOS VIRTUAIS E DE DATAGRAMAS

Lembre-se de que, no Capítulo 3, dissemos que a camada de transporte pode oferecer às aplicações serviço não orientado para conexão ou serviço orientado para conexão. Por exemplo, a camada de transporte da Internet oferece a cada aplicação uma alternativa entre dois serviços: UDP, um não orientado para conexão; ou TCP, orientado para conexão. De modo semelhante, uma camada de rede também pode oferecer qualquer dos dois. Serviços de camada de rede orientados para conexão e não orientados para conexão são, em muitos aspectos, semelhantes a esses mesmos serviços providos pela camada de transporte. Por exemplo, um serviço de camada de rede orientado para conexão começa com uma apresentação entre os hospedeiros de origem e de destino; e um serviço de camada de rede não orientado para conexão não tem nenhuma apresentação preliminar.

Embora os serviços de camada de rede orientados para conexão e não orientados para conexão tenham algumas semelhanças com os mesmos serviços oferecidos pela camada de transporte, há diferenças cruciais:

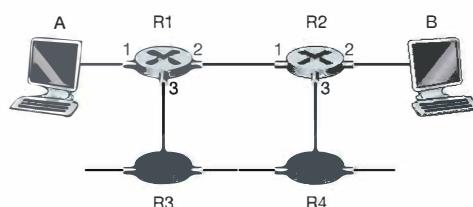
- Na camada de rede, são serviços de hospedeiro a hospedeiro providos pela camada de rede à camada de transporte. Na camada de transporte, são serviços de processo a processo fornecidos pela camada de transporte à camada de aplicação.
- Em todas as arquiteturas importantes de redes de computadores existentes até agora (Internet, ATM, *frame relay* e assim por diante), a camada de rede oferece um serviço entre hospedeiros não orientado para conexão, ou um serviço entre hospedeiros orientado para conexão, mas não ambos. Redes de computadores que oferecem apenas um serviço orientado para conexão na camada de rede são denominadas **redes de circuitos virtuais (redes CV)**; redes de computadores que oferecem apenas um serviço não orientado para conexão na camada de rede são denominadas **redes de datagramas**.
- As execuções de serviço orientado para conexão na camada de transporte e de serviço de conexão na camada de rede são fundamentalmente diferentes. No capítulo anterior vimos que o serviço de camada de transporte orientado para conexão é executado na borda da rede nos sistemas finais; em breve veremos que o serviço da camada de rede orientado para conexão é realizado nos roteadores no núcleo da rede, bem como nos sistemas finais.

Redes de circuitos virtuais e redes de datagramas são duas classes fundamentais de redes de computadores. Elas utilizam informações muito diferentes para tomar suas decisões de repasse. Vamos agora examinar suas implementações mais de perto.

4.2.1 Redes de circuitos virtuais

Embora a Internet seja uma rede de datagramas, muitas arquiteturas de rede alternativas — entre elas as das redes ATM e *frame relay* — são redes de circuitos virtuais e, portanto, usam conexões na camada de rede. Essas conexões de camada de rede são denominadas **circuitos virtuais (CVs)**. Vamos considerar agora como um serviço de CVs pode ser implantado em uma rede de computadores.

Um circuito virtual (CV) consiste em (1) um caminho (isto é, uma série de enlaces e roteadores) entre hospedeiros de origem e de destino, (2) números de CVs, um número para cada enlace ao longo do caminho e (3) registros na tabela de repasse em cada roteador ao longo do caminho. Um pacote que pertence a um circuito virtual portará um número de CV em seu cabeçalho. Como um circuito virtual pode ter um número de CV diferente em cada enlace, cada roteador interveniente deve substituir esse número de cada pacote em trânsito por um novo número. Esse número novo do CV é obtido da tabela de repasse.

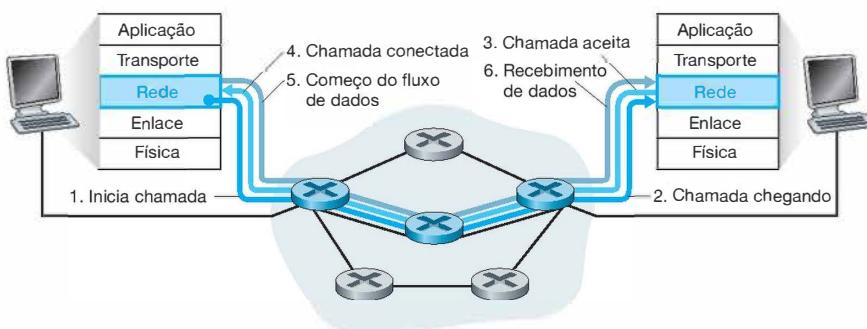
Para ilustrar o conceito, considere a rede mostrada na Figura 4.3. Os números ao lado dos enlaces de R1 na Figura 4.3 são os das interfaces de enlaces. Suponha agora que o hospedeiro A solicite à rede que estabeleça um CV entre ele e o hospedeiro B. Suponha ainda que a rede escolha o caminho A–R1–R2–B e atribua números de CV 12, 22, 32 aos três enlaces no caminho para o circuito virtual. Nesse caso, quando um pacote nesse CV sai do hospedeiro A, o valor no campo do número de CV é 12; quando sai de R1, o valor é 22, e, quando sai de R2, o valor é 32.

Como o roteador determina o novo número de CV para um pacote que passa por ele? Para uma rede de CV, a tabela de repasse de cada roteador inclui a tradução de número de CV; por exemplo, a tabela de repasse de R1 pode ser algo parecido com o seguinte:

Interface de entrada	Nº do CV de entrada	Interface de saída	Nº do CV de saída
1	12	2	22
2	63	1	18
3	7	2	17
1	97	3	87
...	...	...	...

Sempre que um novo CV é estabelecido através de um roteador, um registro é adicionado à tabela de repasse. De maneira semelhante, sempre que um CV termina, são removidos os registros apropriados em cada tabela ao longo de seu caminho.

É bem provável que você esteja pensando por que um pacote não conserva o mesmo número de CV em cada um dos enlaces ao longo de sua rota. Isso ocorre por dois motivos: primeiro, substituir o número de enlace em enlace reduz o comprimento do campo do CV no cabeçalho do pacote. Segundo, e mais importante, o esta-

FIGURA 4.3 UMA REDE DE CIRCUITOS VIRTUAIS SIMPLES

belecimento de um CV é consideravelmente simplificado se for permitido um número diferente de CV em cada enlace no caminho do circuito virtual. De modo específico, com vários números de CV, cada enlace do caminho pode escolher um número de CV independente daqueles escolhidos em outros enlaces do caminho. Se fosse exigido um mesmo número de CV para todos os enlaces, os roteadores teriam de trocar e processar um número substancial de mensagens para escolher o número de CV a ser usado para uma conexão (por exemplo, um número que não está sendo usado por nenhum outro CV nesses roteadores).

Em uma rede de circuitos virtuais, os roteadores da rede devem manter **informação de estado de conexão** para as conexões em curso. Especificamente, cada vez que uma nova conexão for estabelecida através de um roteador, um novo registro de conexão deve ser adicionado à tabela de repasse do roteador. E, sempre que uma conexão for desativada, um registro deve ser removido da tabela. Observe que, mesmo que não haja tradução de números de CVs, ainda assim é necessário manter informação de estado de conexão que associe números de CVs com números das interfaces de saída. A questão de um roteador manter ou não informação de estado de conexão para cada conexão em curso é crucial — e retornaremos a ela várias vezes neste livro.

Há três fases que podem ser identificadas em um circuito virtual:

- *Estabelecimento de CV*. Durante a fase de estabelecimento, a camada de transporte remetente contata a camada de rede, especifica o endereço do receptor e espera até a rede estabelecer o CV. A camada de rede determina o caminho entre remetente e destinatário, ou seja, a série de enlaces e roteadores pelos quais todos os pacotes do CV trafegarão. A camada de rede também determina o número de CV para cada enlace ao longo do caminho e, por fim, adiciona um registro na tabela de repasse em cada roteador no caminho. Durante o estabelecimento do CV, a camada de rede pode também reservar recursos (por exemplo, largura de banda) no caminho.
- *Transferência de dados*. Como mostra a Figura 4.4, tão logo estabelecido o CV, pacotes podem começar a fluir ao longo dele.
- *Encerramento do CV*. O encerramento começa quando o remetente (ou o destinatário) informa à camada de rede seu desejo de desativar o CV. A camada de rede então informará o sistema final do outro lado da rede do término de conexão e atualizará as tabelas de repasse em cada um dos roteadores de pacotes no caminho para indicar que o CV não existe mais.

Há uma distinção sutil, mas importante, entre estabelecimento de CV na camada de rede e estabelecimento de conexão na camada de transporte (por exemplo, a apresentação TCP de três vias que estudamos no Capítulo 3). Estabelecer conexão na camada de transporte envolve apenas os dois sistemas finais. Durante o estabelecimento da conexão na camada de transporte, os dois sistemas finais determinam os parâmetros (por exemplo, número de sequência inicial e tamanho da janela de controle de fluxo) de sua conexão de camada de transporte. Embora os dois sistemas finais fiquem cientes da conexão de camada de transporte, os roteadores dentro da rede

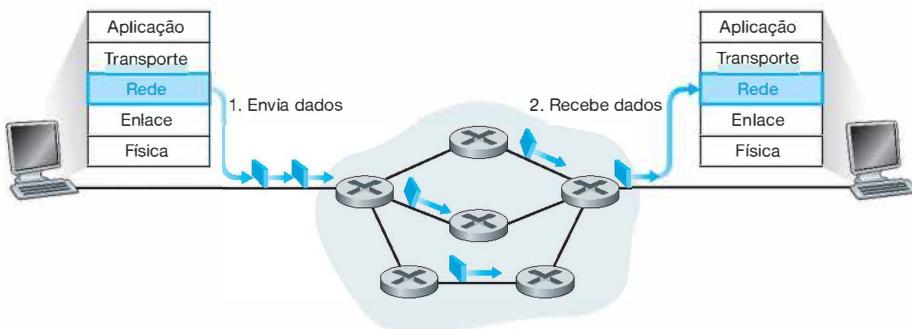
FIGURA 4.4 ESTABELECIMENTO DE CIRCUITO VIRTUAL

Aplicação
Transporte
Rede
Enlace
Física
4. Chamada conectada
5. Começo do fluxo de dados
3. Chamada aceita
6. Recebimento de dados
Aplicação
Transporte
Rede
Enlace
Física
1. Inicia chamada
2. Chamada chegando

ficam completamente alheios a ela. Por outro lado, com uma camada de rede de CV, *os roteadores do caminho entre os dois sistemas finais estão envolvidos no estabelecimento de CV e cada roteador fica totalmente ciente de todos os CVs que passam por ele.*

As mensagens que os sistemas finais enviam à rede para iniciar ou encerrar um CV e aquelas passadas entre os roteadores para estabelecer o CV (isto é, modificar estado de conexão em tabelas de roteadores) são conhecidas como **mensagens de sinalização** e os protocolos usados para trocá-las costumam ser denominados **protocolos de sinalização**. O estabelecimento de CV está ilustrado na Figura 4.4. Não abordaremos protocolos de sinalização de CVs neste livro; Black [1997] apresenta uma discussão geral sobre sinalização em redes orientadas para conexão e ITU-T Q.2931 [1994] mostra a especificação do protocolo de sinalização Q.2931 do ATM.

4.2.2 Redes de datagramas

Em uma **rede de datagramas**, toda vez que um sistema final quer enviar um pacote, ele marca o pacote com o endereço do sistema final de destino e então o envia para dentro da rede. Como mostra a Figura 4.5, isso é feito sem o estabelecimento de nenhum CV. Roteadores em uma rede de datagramas não mantêm nenhuma informação de estado sobre CVs (porque não há nenhum!).

Ao ser transmitido da origem ao destino, um pacote passa por uma série de roteadores. Cada um desses roteadores usa o endereço de destino do pacote para repassá-lo. Especificamente, cada roteador tem uma tabela de repasse que mapeia endereços de destino para interfaces de enlaces; quando um pacote chega ao roteador, este usa o endereço de destino do pacote para procurar a interface de enlace de saída apropriada na tabela de repasse. Então, o roteador transmite o pacote para aquela interface de enlace de saída.

Para entender melhor a operação de consulta, vamos examinar um exemplo específico. Suponha que todos os endereços de destino tenham 32 bits (que, por acaso, é exatamente o comprimento do endereço de destino em um datagrama IP). Uma execução de força bruta da tabela de repasse teria um registro para cada endereço de destino possível. Como há mais de quatro bilhões de endereços possíveis, essa opção está totalmente fora de questão.

Vamos supor ainda que nosso roteador tenha quatro enlaces numerados de 0 a 3, e que os pacotes devem ser repassados para as interfaces de enlace como mostrado a seguir:

Faixa de endereços de destino	Interface de enlace
11001000 00010111 00010000 00000000 até 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 até 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 até 11001000 00010111 00011111 11111111	2
senão	3

FIGURA 4.5 REDE DE DATAGRAMAS

Fica claro, por este exemplo, que não é necessário ter quatro bilhões de registros na tabela de repasse do roteador. Poderíamos, por exemplo, ter a seguinte tabela de repasse com apenas quatro registros:

Prefixo do endereço	Interface de enlace
11001000 00010111 00010	0
11001000 00010111 00011000	1
11001000 00010111 00011	2
senão	3

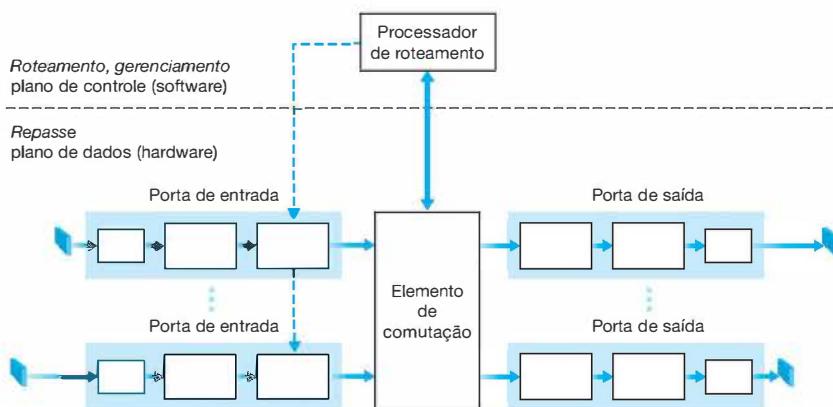
Com esse tipo de tabela de repasse, o roteador compara um **prefixo** do endereço de destino do pacote com os registros na tabela; se houver uma concordância de prefixos, o roteador transmite o pacote para o enlace associado àquele prefixo correspondente. Por exemplo, suponha que o endereço de destino do pacote seja 11001000 00010111 00010110 10100001; como o prefixo de 21 bits desse endereço é igual ao primeiro registro na tabela, o roteador transmite o pacote para a interface de enlace 0. Se o prefixo do pacote não combinar com nenhum dos três primeiros registros, o roteador envia o pacote para a interface 3. Embora isso pareça bastante simples, há aqui uma sutileza importante. Você talvez tenha notado a possibilidade de um endereço de destino combinar com mais de um registro. Por exemplo, os primeiros 24 bits do endereço 11001000 00010111 00011000 10101010 combinam com o segundo registro na tabela e os primeiros 21 bits do endereço combinam com o terceiro registro. Quando há várias concordâncias de prefixos, o roteador usa a **regra da concordância do prefixo mais longo**, isto é, encontra o registro cujo prefixo tem mais bits correspondentes aos bits do endereço do pacote e envia o pacote à interface de enlace associada com esse prefixo mais longo que tenha correspondência. Veremos exatamente por que essa regra de prefixo mais longo correspondente é utilizada quando estudarmos endereçamento da Internet em mais detalhes da Seção 4.4.

Embora em redes de datagramas os roteadores não mantenham nenhuma informação de estado de conexão, ainda assim mantêm informação de estado de repasse em suas tabelas de repasse. Todavia, a escala temporal da mudança dessas informações de estado é um tanto lenta. Na verdade, as tabelas de repasse em uma rede de datagramas são modificadas pelos algoritmos de roteamento que em geral atualizam uma tabela de repasse em intervalos de um a cinco minutos, mais ou menos. A tabela de repasse de um roteador em uma rede de CVs é modificada sempre que é estabelecida uma nova conexão através do roteador ou sempre que uma conexão existente é desativada. Em um roteador de *backbone* de nível 1, isso poderia acontecer facilmente em uma escala temporal de microssegundos.

Como em redes de datagramas as tabelas de repasse podem ser modificadas a qualquer momento, uma série de pacotes enviados de um sistema final para outro pode seguir caminhos diferentes pela rede e muitos podem chegar fora da ordem. Paxson [1997] e Jaiswal [2003] apresentam interessantes estudos de medição da reordenação de pacotes e de outros fenômenos na Internet pública.

4.2.3 Origens das redes de circuitos virtuais e de datagramas

A evolução das redes de datagramas e de circuitos virtuais reflete as origens dessas redes. A ideia de um circuito virtual como princípio fundamental de organização tem suas raízes no mundo da telefonia (que usa circuitos reais). Como em redes de CV os roteadores mantêm estabelecimento da chamada e estado por chamada, esse tipo de rede é consideravelmente mais complexo do que uma rede de datagramas. (Molinero-Fernandez [2002] faz uma comparação interessante entre as complexidades de redes de comutação de circuitos e de comutação de pacotes.) Isso também está de acordo com a herança da telefonia. A complexidade das redes telefônicas estava, necessariamente, dentro da própria rede, já que elas conectavam sistemas finais não inteligentes tais como telefones de disco. (Para os mais jovens que não o conhecem, um telefone de disco é um telefone analógico [isto é, não digital] sem teclas — tem somente um mostrador com números e um dispositivo mecânico denominado disco.)

Por sua vez, a Internet como uma rede de datagramas surgiu da necessidade de conectar computadores. Como esses sistemas finais são mais sofisticados, os arquitetos da Internet preferiram construir um modelo de serviço de camada de rede o mais simples possível. Como já vimos nos capítulos 2 e 3, funcionalidades adicionais (por exemplo, entrega na ordem, transferência confiável de dados, controle de congestionamento e resolução de nomes DNS) são executadas em uma camada mais alta, nos sistemas finais. Isso inverte o modelo da rede de telefonia, com algumas consequências interessantes:

- Visto que o modelo de serviço de camada de rede resultante da Internet, que oferece garantias mínimas (nenhuma!) de serviço, impõe exigências mínimas sobre a camada de rede. Isso facilita a interconexão de redes que usam tecnologias de camada de enlace muito diferentes (por exemplo, satélite, Ethernet, fibra ou rádio) e cujas taxas de transmissão e características de perda também são muito diferentes. Vamos abordar detalhadamente a interconexão de redes IP na Seção 4.4.
- Como vimos no Capítulo 2, aplicações como e-mail, a Web e até mesmo um serviço centrado na camada de rede como o DNS são executadas em hospedeiros (servidores) na borda da rede. A capacidade de adicionar um novo serviço apenas ligando um hospedeiro à rede e definindo um novo protocolo de camada de aplicação (como o HTTP) permitiu que novas aplicações como a Web fossem distribuídas pela Internet em um período notavelmente curto.

4.3 O QUE HÁ DENTRO DE UM ROTEADOR?

Agora que já tivemos uma visão geral das funções e serviços da camada de rede, voltaremos nossa atenção para a **função de repasse** — a transferência, propriamente dita, de pacotes dos enlaces de entrada até os enlaces de saída adequados de um roteador. Já estudamos algumas questões de repasse na Seção 4.2, a saber, endereçamento e correspondência com o prefixo mais longo. Mencionamos, de passagem, que os pesquisadores e profissionais de redes de computadores usam as palavras *repasse* e *comutação* indistintamente; nós usaremos ambos os termos neste livro.

Uma visão de alto nível da arquitetura de um roteador genérico é mostrada na Figura 4.6. Quatro componentes de um roteador podem ser identificados:

- *Portas de entrada*. A porta de entrada tem diversas funções. Ela realiza as funções de camada física (a caixa mais à esquerda da porta de entrada e a caixa mais à direita da porta de saída na Figura 4.6) de terminar um enlace físico de entrada em um roteador. Executa também as de camada de enlace (representadas pelas caixas do meio nas portas de entrada e de saída) necessárias para interoperar com as funções da camada de enlace do outro lado do enlace de entrada. Talvez mais importante, a função de exame também é realizada na porta de entrada; isso ocorrerá na caixa mais à direita da porta de entrada. É aqui que a tabela de repasse é consultada para determinar a porta de saída do roteador à qual um pacote que chega será repassado por meio do elemento de comutação. Pacotes de controle (por exemplo, pacotes carregando informações de protocolo de roteamento) são repassados de uma porta de entrada até o

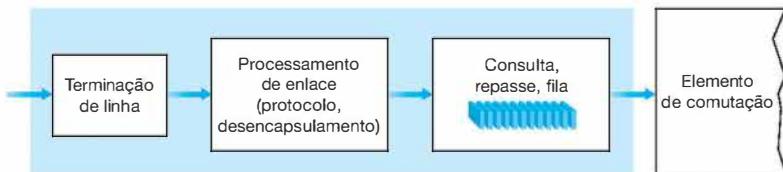
FIGURA 4.6 ARQUITETURA DE ROTEADOR

processador de roteamento. Note que o termo *porta* aqui — referindo-se às interfaces físicas de entrada e saída do roteador — é distintamente diferente das portas de software associadas a aplicações de rede e *sockets*, discutidos nos Capítulos 2 e 3.

- *Elemento de comutação.* O elemento de comutação conecta as portas de entrada do roteador às suas portas de saída. Ele está integralmente contido no interior do roteador — uma rede dentro de um roteador da rede!
- *Portas de saída.* Uma porta de saída armazena os pacotes que foram repassados a ela através do elemento de comutação e, então, os transmite até o enlace de saída, realizando as funções necessárias da camada de enlace e da camada física. Quando um enlace é bidirecional (isto é, carrega um tráfego em ambas as direções), uma porta de saída para o enlace será emparelhada com a porta de entrada para esse enlace na mesma placa de linha (uma placa de circuito impresso contendo uma ou mais portas de entrada, e que está conectada ao elemento de comutação).
- *Processador de roteamento.* O processador de roteamento executa os protocolos de roteamento (que estudaremos na Seção 4.6), mantém as tabelas de roteamento e as informações de estado do enlace, e calcula a tabela de repasse para o roteador. Ele também realiza funções de gerenciamento de rede, que estudaremos no Capítulo 9.

Lembre-se de que, na Seção 4.1.1, distinguimos entre as funções de repasse e roteamento de um roteador. As portas de entrada, portas de saída e elemento de comutação de um roteador executam a função de repasse e quase sempre são implementadas no hardware, como ilustra a Figura 4.6. Essas funções às vezes são chamadas coletivamente de **plano de repasse do roteador**. Para entender por que é necessário haver uma execução no hardware, considere que, com um enlace de entrada de 10 bits/s e um datagrama IP de 64 bytes, a porta de entrada tem apenas 51,2 ns para processar o datagrama antes que outro datagrama possa chegar. Se *N* portas forem combinadas em uma placa de linha (como em geral é feito na prática), a canalização de processamento de datagrama precisa operar *N* vezes mais rápido — muito rápido para uma realização em software. O hardware do plano de repasse pode ser realizado usando os próprios projetos de hardware de um fabricante de roteador ou pode ser construído usando chips de silício comprados no mercado (por exemplo, vendidos por empresas como Intel e Broadcom).

Embora o plano de repasse opere em uma escala de tempo de nanossegundo, as funções de controle de um roteador — executando os protocolos de roteamento, respondendo a enlaces conectados que são ativados ou desativados, e realizando funções de gerenciamento como aquelas que estudaremos no Capítulo 9 — operam na escala de tempo de milissegundo ou segundo. Essas funções do **plano de controle do roteador** costumam ser realizadas no software e executam no processador de roteamento (em geral, uma CPU tradicional).

Antes de entrarmos nos detalhes do plano de controle e dados de um roteador, vamos retornar à analogia da Seção 4.1.1, em que o repasse de pacotes foi comparado com carros entrando e saindo de um pedágio. Vamos supor que, antes que um carro entre no pedágio, algum processamento seja necessário — o veículo para em uma estação de entrada e indica seu destino final (não no pedágio local, mas o destino final de sua viagem). Um atendente na cabine examina o destino final, determina a saída do pedágio que leva a esse destino final e diz ao motorista qual saída ele deve tomar. O carro entra no pedágio (que pode estar cheio de outros carros entrando de outras estradas de entrada e seguindo para outras saídas do pedágio) e por fim segue pela pista de saída indicada, onde poderá encontrar outros carros saindo do pedágio nessa mesma saída.

Nessa analogia, podemos reconhecer os componentes principais do roteador na Figura 4.6 — a pista de entrada e a cabine de entrada correspondem à porta de entrada (com uma função de consulta para determinar a porta de saída local); o pedágio corresponde ao elemento de comutação; e a pista de saída do pedágio corresponde à porta de saída. Com essa analogia, é instrutivo considerar onde poderiam acontecer os gargalos. O que acontece se os carros chegarem muito depressa (por exemplo, o pedágio está na Alemanha ou na Itália!), mas o atendente na cabine for lento? Com que velocidade o atendente deverá trabalhar para garantir que não haja engarrafamento na pista de entrada? Mesmo com um atendente incrivelmente rápido, o que acontece se os carros atravessarem o pedágio devagar — ainda poderá haver engarrafamentos? E o que ocorre se a maioria dos carros que entram quiserem sair do pedágio na mesma pista de saída — pode haver engarrafamentos na pista de saída ou em outro lugar? Como o pedágio deve ser operado se quisermos atribuir prioridades a carros diferentes, ou impedir que certos veículos sequer entrem no pedágio? Todas estas são questões críticas semelhantes às enfrentadas pelos projetistas de roteador e de comutador.

Nas próximas subseções, vamos examinar as funções do roteador com mais detalhes. Iyer [2008]; Chao [2001]; Chuang [2005]; Turner [1988]; McKeown [1997a] e Partridge [1998] oferecem uma discussão das arquiteturas específicas de roteador. Por concretude, a discussão a seguir considera uma rede de datagramas em que as decisões de repasse são baseadas no endereço de destino do pacote (em vez de um número de VC em uma rede de circuitos virtuais). Porém, os conceitos e as técnicas são muito semelhantes para uma rede de circuitos virtuais.

4.3.1 Processamento de entrada

Uma visão mais detalhada da funcionalidade de porta da entrada é apresentada na Figura 4.7. Como discutido anteriormente, as funções de terminação de linha e de processamento de enlace realizadas pela porta de entrada implementam as funções das camadas física e de enlace associadas a um enlace de entrada individual do roteador. A pesquisa realizada na porta de entrada é fundamental para a operação do roteador — é aqui que o roteador usa a tabela de repasse para determinar a porta de saída para a qual o pacote que está chegando será repassado pelo elemento de comutação. A tabela de repasse é calculada e atualizada pelo processador de roteamento, e uma cópia da tabela é comumente armazenada em cada porta de entrada. A tabela de repasse é copiada do processador de roteamento para as placas de linha por um barramento separado (por exemplo, um barramento PCI), indicado na Figura 4.6 pela linha tracejada do processador de roteamento às placas de linha da entrada. Com uma cópia de sombra, as decisões de repasse podem ser feitas no local, em cada porta de entrada, sem chamada ao processador de roteamento centralizado a cada pacote, evitando assim um gargalo de processamento centralizado.

Dada a existência de uma tabela de repasse, o exame é conceitualmente simples — basta procurar o registro mais longo correspondente ao endereço de destino, como descrito na Seção 4.2.2. Porém, com taxas de transmissão de gigabits, a procura precisa ser realizada em nanossegundos (lembre-se do exemplo anterior de um enlace de 10 bits/s e um datagrama IP de 64 bytes). Assim, não apenas a pesquisa deve ser realizada no hardware, mas são necessárias outras técnicas além da busca linear simples por uma tabela grande; estudos sobre algoritmos de pesquisa rápidos podem ser encontrados em Gupta [2001] e Ruiz-Sanchez [2001]. É preciso prestar atenção especial aos tempos de acesso da memória, resultando em projetos com memórias de DRAM e SRAM (usadas como cache de DRAM) mais rápidas, embutidas no chip. *Ternary Content Address Memories* (TCAMs) também

FIGURA 4.7 PROCESSAMENTO NA PORTA DE ENTRADA

são usadas para pesquisa. Com uma TCAM, um endereço IP de 32 bits é apresentado à memória, que retorna o conteúdo da entrada da tabela de repasse para esse endereço em um tempo constante. O Cisco 8500 tem uma CAM de 64K para cada porta de entrada.

Quando a porta de saída de um pacote tiver sido determinada por meio da pesquisa, ele pode ser enviado para o elemento de comutação. Em alguns projetos, um pacote pode ser temporariamente impedido de entrar no elemento de comutação se os pacotes de outras portas de entrada estiverem usando o elemento nesse instante. Um pacote impedido ficará enfileirado na porta de entrada e depois escalonado para cruzar o elemento em outra oportunidade. Veremos mais de perto os processos de bloqueio, enfileiramento e escalonamento de pacotes (nas portas de entrada e de saída) na Seção 4.3.4. Embora a "pesquisa" seja comprovadamente a ação mais importante no processamento da porta de entrada, muitas outras ações devem ser tomadas: (1) o processamento da camada física e de enlace deverá ocorrer, conforme já vimos; (2) os campos de número de versão, soma de verificação e tempo de vida do pacote — todos eles estudados na Seção 4.4.1 — deverão ser verificados, e os dois últimos campos reescritos; e (3) contadores usados para o gerenciamento de rede (como o número de datagramas IP recebidos) devem ser atualizados.

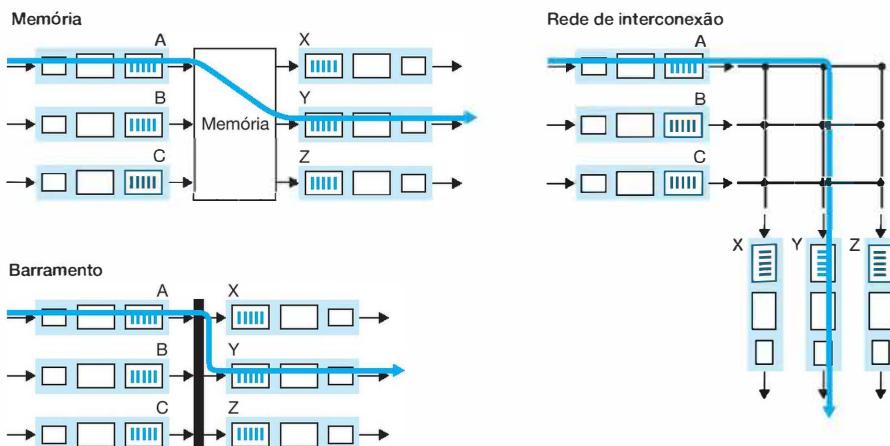
Vamos encerrar nossa discussão de processamento de porta de entrada observando que as etapas da porta de entrada de pesquisar um endereço IP ("combinação") e depois enviar o pacote para o elemento de comutação ("ação") é um caso específico de uma abstração "combinação mais ação", mais ampla, que é realizada em muitos

HISTÓRIA

Dominando o núcleo da rede

Quando este livro foi escrito, em 2012, a Cisco empregava mais de 65 mil pessoas. Como surgiu essa gigantesca empresa de rede? Tudo começou em 1984 na sala de estar de um apartamento no Vale do Silício.

Len Bosak e sua esposa Sandy Lerner trabalhavam na universidade de Stanford quando tiveram a ideia de construir e vender roteadores de Internet a instituições acadêmicas e de pesquisa, os primeiros a adotarem a Internet naquela época. Sandy Lerner teve a ideia do nome "Cisco" (uma abreviação de San Francisco) e também criou o logotipo da empresa, que é uma ponte. A sede era originalmente a sala de estar do casal que, no início, financiou o projeto com cartões de crédito e trabalhos de consultoria à noite. No final de 1986, as receitas da empresa chegaram a 250 mil dólares por mês. No fim de 1987, a Cisco conseguiu atrair capital de risco de dois milhões de dólares da Sequoia Capital em troca de um terço da empresa. Nos anos seguintes continuou a crescer e a conquistar cada vez mais participação de mercado. Ao mesmo tempo, o relacionamento entre Bosak/Lerner e a administração da empresa começou a ficar tenso. A Cisco abriu seu capital em 1990 e, nesse mesmo ano, Lerner e Bosak saíram da empresa.

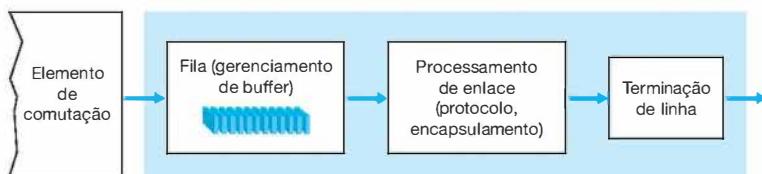
Com o passar dos anos, a Cisco expandiu amplamente seu mercado de roteadores, vendendo produtos voltados à segurança, redes sem fio, comutador Ethernet, infraestrutura de centro de dados, videoconferência e produtos e serviços de Voz sobre IP. Entretanto, está enfrentando concorrência internacional crescente, incluindo a empresa chinesa em crescimento Huawei, que fornece equipamentos de rede. Outras fontes de concorrência para a Cisco no mercado de roteadores e Ethernet comutada incluem a Alcatel-Lucent e a Juniper.

dispositivos da rede, não apenas roteadores. Em comutadores da camada de enlace (explicados no Capítulo 5), os endereços de destino da camada de enlace são pesquisados e várias ações podem ser tomadas além do envio do quadro ao elemento de comutação pela porta de saída. Em *firewalls* (explicados no Capítulo 8) — dispositivos que filtram pacotes selecionados que chegam —, um pacote que chega, cujo cabeçalho combina com determinado critério (por exemplo, uma combinação de endereços IP de origem/destino e números de porta da camada de transporte), pode ter o repasse impedido (ação). Em um tradutor de endereço de rede (NAT, *Network Address Translator*, discutido na Seção 4.4), um pacote que chega, cujo número de porta da camada de transporte combina com determinado valor, terá seu número de porta reescrito antes de ser repassado (ação). Assim, a abstração "combinação mais ação" é tanto poderosa quanto prevalente nos dispositivos da rede.

4.3.2 Elemento de comutação

O elemento de comutação está no coração de um roteador. É por meio do elemento de comutação que os pacotes são comutados (isto é, repassados) de uma porta de entrada para uma porta de saída. A comutação pode ser realizada de inúmeras maneiras, como mostra a Figura 4.8.

- *Comutação por memória.* Os primeiros e mais simples roteadores quase sempre eram computadores tradicionais nos quais a comutação entre as portas de entrada e de saída era realizada sob o controle direto da CPU (processador de roteamento). Essas portas funcionavam como dispositivos tradicionais de entrada/saída de um sistema operacional tradicional. Uma porta de entrada na qual um pacote estivesse entrando primeiro sinalizaria ao processador de roteamento por meio de uma interrupção. O pacote era então copiado da porta de entrada para a memória do processador. O processador de roteamento então extraía o endereço de destino do cabeçalho, consultava a porta de saída apropriada na tabela de repasse e copiava o pacote para os buffers da porta de saída. Neste cenário, se a largura de banda da memória for tal que *B* pacotes/segundo possam ser escritos ou lidos na memória, então a vazão total de repasse (a velocidade total com que os pacotes são transferidos de portas de entrada para portas de saída) deverá

FIGURA 4.8 TRÊS TÉCNICAS DE COMUTAÇÃO

ser menor do que $B/2$. Observe também que dois pacotes não podem ser repassados ao mesmo tempo, mesmo que tenham diferentes portas de destino, pois somente uma leitura/escrita de memória pelo barramento compartilhado do sistema pode ser feita de cada vez.

Muitos roteadores modernos também comutam por memória. Contudo, uma diferença importante entre esses roteadores e os antigos é que a consulta do endereço de destino e o armazenamento do pacote na localização adequada da memória são realizados por processadores nas placas de linha de entrada. Em certos aspectos, roteadores que comutam por memória se parecem muito com multiprocessadores de memória compartilhada, nos quais os processadores de uma placa de linha comutam (escrevem) pacotes para a memória da porta de saída adequada. Os comutadores série 8500 Catalyst da Cisco [Cisco 8500, 2012] comutam pacotes por uma memória compartilhada.

- *Comutação por um barramento.* Nessa abordagem, as portas de entrada transferem um pacote diretamente para a porta de saída por um barramento compartilhado sem a intervenção do processador de roteamento. Para isso, a porta de entrada insere um rótulo interno ao comutador (cabeçalho) antes do pacote, indicando a porta de saída local à qual ele está sendo transferido e o pacote é transmitido para o barramento. Ele é recebido por todas as portas de saída, mas somente a porta que combina com o rótulo manterá o pacote. O rótulo é então removido na porta de saída, pois só é usado dentro do comutador para atravessar o barramento. Se vários pacotes chegarem ao roteador ao mesmo tempo, cada um em uma porta de entrada diferente, todos menos um deverão esperar, pois apenas um pacote pode cruzar o barramento de cada vez. Como cada pacote precisa atravessar o único barramento, a velocidade de comutação do roteador é limitada à velocidade do barramento; em nossa analogia com o sistema de pedágio, é como se o sistema de pedágio só pudesse conter um carro de cada vez. Apesar disso, a comutação por um barramento muitas vezes é suficiente para roteadores que operam em redes de acesso e redes de empresas. Os comutadores Cisco 5600 [Cisco Switches, 2012] comutam pacotes por um barramento da placa-mãe (*backplane bus*) de 32 bits/s.
- *Comutação por uma rede de interconexão.* Um modo de vencer a limitação da largura de banda de um barramento único compartilhado é usar uma rede de interconexão mais sofisticada, tal como as que eram utilizadas no passado para interconectar processadores em uma arquitetura de computadores multiprocessadores. Um comutador do tipo *crossbar* é uma rede de interconexão que consiste em $2n$ barramentos que conectam n portas de entrada com n portas de saída, como ilustra a Figura 4.8. Cada barramento vertical atravessa cada barramento horizontal em um cruzamento, que pode ser aberto ou fechado a qualquer momento pelo controlador do elemento de comutação (cuja lógica faz parte do próprio elemento de comutação). Quando um pacote chega da porta A e precisa ser repassado para a porta Y, o controlador do comutador fecha o cruzamento na interseção dos barramentos A e Y, e a porta A, então, envia o pacote para seu barramento, que é apanhado (apenas) pelo barramento Y. Observe que um pacote da porta B pode ser repassado para a porta X ao mesmo tempo, pois os pacotes A-para-Y e B-para-X usam diferentes barramentos de entrada e saída. Assim, diferentemente das duas técnicas de comutação anteriores, as redes do tipo *crossbar* são capazes de repassar vários pacotes em paralelo. Porém, se dois pacotes de duas portas de entrada diferentes forem destinados à mesma porta de saída, então um terá que esperar na entrada, pois somente um pacote pode ser enviado por qualquer barramento de cada vez.

  Redes de comutação mais sofisticadas utilizam vários estágios de elementos de comutação para permitir que pacotes de diferentes portas de entrada prossigam para a mesma porta de saída ao mesmo tempo através do elemento de comutação. Consulte Tobagi [1990] para ver um levantamento de arquiteturas de comutação. Os comutadores da família 12000 da Cisco [Cisco 12000, 2012] usam uma rede de interconexão.

4.3.3 Processamento de saída

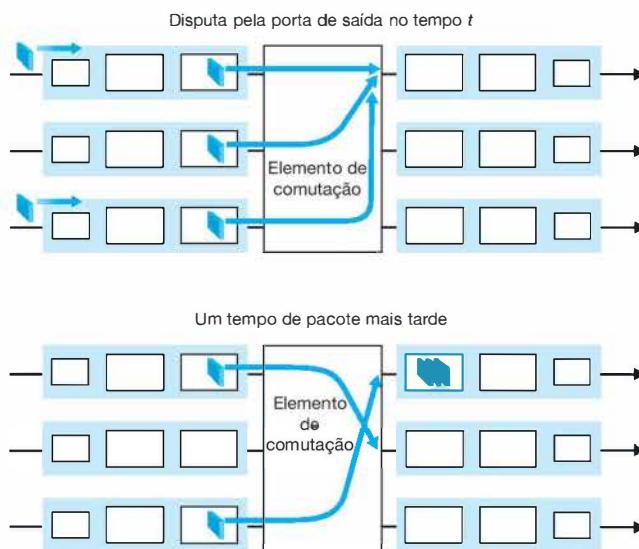
O processamento de portas de saída, mostrado na Figura 4.9, toma os pacotes que foram armazenados na memória da porta de saída e os transmite pelo enlace de saída. Isso inclui a seleção e a retirada dos pacotes da fila para transmissão, com a realização das funções de transmissão necessárias nas camadas de enlace e física.

FIGURA 4.9 PROCESSAMENTO DE PORTA DE SAÍDA

4.3.4 Onde ocorre formação de fila?

Se examinarmos a funcionalidade da porta de entrada e da porta de saída e as configurações mostradas na Figura 4.8, veremos que filas de pacotes podem se formar tanto nas portas de entrada como nas de saída, assim como identificamos casos em que os carros podem esperar nas entradas e saídas em nossa analogia de pedágio. O local e a extensão da formação de fila (seja nas da porta de entrada ou nas da porta de saída) dependerão da carga de tráfego, da velocidade relativa do elemento de comutação e da taxa da linha. Agora, vamos examinar essas filas com um pouco mais de detalhes, já que, à medida que elas ficam maiores, a memória do roteador será finalmente exaurida e ocorrerá **perda de pacote**, quando nenhuma memória estará disponível para armazenar os pacotes que chegam. Lembre-se de que, em nossas discussões anteriores, dissemos que pacotes eram "perdidos dentro da rede" ou "descartados em um roteador". E é aí, nessas filas dentro de um roteador, que esses pacotes são de fato descartados e perdidos.

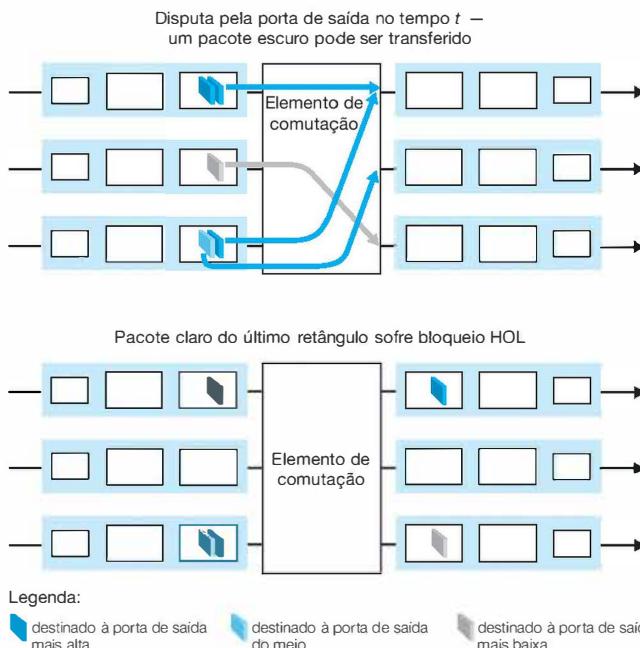
Suponha que as taxas da linha de entrada e as taxas da linha de saída (taxas de transmissão) tenham todas uma taxa de transmissão idêntica de R_{linha} pacotes por segundo, e que haja N portas de entrada e N portas de saída. Para simplificar ainda mais a discussão, vamos supor que todos os pacotes tenham o mesmo comprimento fixo e que eles chegam às portas de entrada de uma forma síncrona. Isto é, o tempo para enviar um pacote em qualquer enlace é igual ao tempo para receber um pacote em qualquer enlace, e durante esse intervalo, zero ou um pacote pode chegar em um enlace de entrada. Defina a taxa de transferência do elemento de comutação $R_{comutação}$ como a taxa na qual os pacotes podem ser movimentados da porta de entrada à porta de saída. Se $R_{comutação}$ for N vezes mais rápida que R_{linha}, então haverá apenas uma formação de fila insignificante nas portas de entrada. Isso porque, mesmo no pior caso, em que todas as N linhas de entrada estiverem recebendo pacotes, e todos eles tiverem de ser repassados para a mesma porta de saída, cada lote de N pacotes (um pacote por porta de entrada) poderá ser absorvido pelo elemento de comutação antes que o próximo lote chegue.

Mas o que pode acontecer nas portas de saída? Vamos supor que $R_{comutação}$ ainda seja N vezes R_{linha}. Mais uma vez, os pacotes que chegarem a cada uma das N portas de entrada serão destinados à mesma porta de saída. Nesse caso, no tempo que leva para enviar um único pacote no enlace de saída, N pacotes novos chegarão a essa porta de saída. Uma vez que essa porta pode transmitir somente um único pacote em cada unidade de tempo (o tempo de transmissão do pacote), os N pacotes que chegarem terão de entrar na fila (esperar) para transmissão pelo enlace de saída. Então, mais N pacotes poderão chegar durante o tempo que leva para transmitir apenas um dos N pacotes que estavam na fila antes, e assim por diante. Por fim, o número de pacotes na fila pode ficar grande o bastante para exaurir o espaço de memória na porta de saída, caso em que os pacotes são descartados.

A formação de fila na porta de saída está ilustrada na Figura 4.10. No tempo t, um pacote chegou a cada uma das portas de entrada, cada um deles destinado à porta de saída que está mais acima na figura. Admitindo

taxas da linha idênticas e um comutador operando a uma taxa três vezes maior do que a da linha, uma unidade de tempo mais tarde (isto é, no tempo necessário para receber ou enviar um pacote), todos os três pacotes originais foram transferidos para a porta de saída e estão em fila aguardando transmissão. Na unidade de tempo seguinte, um desses três terá sido transmitido pelo enlace de saída. Em nosso exemplo, dois *novos* pacotes chegaram do lado de entrada do comutador; um deles é destinado àquela mesma porta de saída que está mais acima na figura.

Dado que os buffers do roteador são necessários para absorver as oscilações da carga de tráfego, deve-se perguntar *quanto* de armazenamento em buffers é necessário. Durante muitos anos, a regra prática [RFC 3439] para dimensionamento de buffers foi que a quantidade de armazenamento em buffers (B) deveria ser igual a um tempo de viagem de ida e volta (RTT, digamos, 250 ms) vezes a capacidade do enlace (C). Esse resultado é baseado em uma análise da dinâmica de filas com um número relativamente pequeno dos fluxos do TCP [Villamizar, 1994]. Assim, um enlace de 10 bits/s com um RTT de 250 ms precisaria de uma quantidade de armazenamento em buffers $B = RTT \cdot C$ = 2,5 bits/s de buffers. Esforços teóricos e experimentais recentes [Appenzeller, 2004], entretanto, sugerem que quando há um grande número de fluxos do TCP (N) passando por um enlace, a quantidade de armazenamento em buffers necessária é $B = RTT \cdot C/\sqrt{N}$. Com um grande número de fluxos passando normalmente por grandes enlaces dos roteadores de *backbone* (consulte, por exemplo, Fraleigh [2003]), o valor de N pode ser grande, e a diminuição do tamanho do buffers necessário se torna bastante significativa. Appenzellar [2004]; Wischik [2005] e Behesht [2008] apresentam discussões esclarecedoras em relação ao problema do dimensionamento de buffers a partir de um ponto de vista teórico, de aplicação e operacional.

Uma consequência da fila na porta de saída é que um **escalonador de pacotes** na porta de saída deve escolher para transmissão um dentre os que estão na fila. Essa seleção pode ser feita com base em uma regra simples, como no escalonamento do primeiro a chegar, primeiro a ser atendido (FCFS), ou por uma regra mais sofisticada, tal como a fila ponderada justa (*weighted fair queuing* — WFQ), que compartilha o enlace de saída com equidade entre as diferentes conexões fim a fim que têm pacotes na fila para transmissão. O escalonamento de pacotes desempenha um papel crucial no fornecimento de **garantia de qualidade de serviço**. Examinaremos o escalonamento de pacotes em profundidade no Capítulo 7. Uma discussão sobre disciplinas de escalonamento de pacotes na porta de saída pode ser encontrada em Cisco Queue [2012].

FIGURA 4.10 FORMAÇÃO DE FILA NA PORTA DE SAÍDA

De modo semelhante, se não houver memória suficiente para armazenar um pacote que está chegando, será preciso tomar a decisão de descartar esse pacote (política conhecida como **descarte do final da fila**) ou remover um ou mais já enfileirados para liberar lugar para o pacote recém-chegado. Em alguns casos pode ser vantajoso descartar um pacote (ou marcar o seu cabeçalho) *antes* de o buffers ficar cheio, para dar um sinal de congestionamento ao remetente. Várias políticas de descarte e marcação de pacotes (conhecidas coletivamente como algoritmos de **gerenciamento ativo de fila** [*active queue management* — AQM]) foram propostas e analisadas [Labrador, 1999; Hollot, 2002]. Um dos algoritmos AQM mais estudados e executados é o de **detecção aleatória rápida** (*random early detection* — RED). Ele mantém uma média ponderada do comprimento da fila de saída. Se o comprimento médio da fila for menor do que um valor limite mínimo, min_{th}, quando um pacote chegar, será admitido na fila. Inversamente, se a fila estiver cheia ou se o comprimento médio da fila for maior do que um valor limite máximo, max_{th}, quando um pacote chegar, será marcado ou descartado. Finalmente, se o pacote chegar e encontrar uma fila de comprimento médio no intervalo [min_{th}, max_{th}], o pacote será marcado ou descartado com uma probabilidade que em geral é alguma função do comprimento médio da fila, de min_{th} e de max_{th}. Foram propostas inúmeras funções probabilísticas para marcação/descarte e várias versões do RED foram modeladas, simuladas e/ou implementadas analiticamente. Christiansen [2001] e Floyd [2012] oferecem visões gerais e indicações de leituras adicionais.

Se o elemento de comutação não for veloz o suficiente (em relação às taxas da linha de entrada) para transmitir sem atraso *todos* os pacotes que chegam através dele, então poderá haver formação de fila também nas portas de entrada, pois os pacotes devem se juntar às filas nas portas de entrada para esperar sua vez de ser transferidos pelo elemento de comutação até a porta de saída. Para ilustrar uma importante consequência dessa fila, considere um elemento de comutação do tipo *crossbar* e suponha que (1) todas as velocidades de enlace sejam idênticas, (2) um pacote possa ser transferido de qualquer uma das portas de entrada até uma dada porta de saída no mesmo tempo que leva para um pacote ser recebido em um enlace de entrada e (3) pacotes sejam movimentados de uma fila de entrada até sua fila de saída desejada no modo FCFS. Vários pacotes podem ser transferidos em paralelo, contanto que suas portas de saída sejam diferentes. Entretanto, se dois pacotes que estão à frente das duas filas de entrada forem destinados à mesma fila de saída, então um deles ficará bloqueado e terá de esperar na fila de entrada — o elemento comutador só pode transferir um pacote por vez até uma porta de saída.

A parte superior da Figura 4.11 apresenta um exemplo em que dois pacotes (mais escuros) à frente de suas filas de entrada são destinados à mesma porta de saída mais alta à direita. Suponha que o elemento de comutação escolha transferir o pacote que está à frente da fila mais alta à esquerda. Nesse caso, o pacote mais escuro na fila mais baixa à esquerda tem de esperar. Mas não é apenas este último que tem de aguardar; também tem de esperar o pacote claro que está na fila atrás dele (no retângulo inferior à esquerda), mesmo que não haja *nenhuma* disputa pela porta de saída do meio à direita (que é o destino do pacote claro). Esse fenômeno é conhecido como **bloqueio de cabeça de fila** (**HOL** — *head-of-the-line blocking*) em um comutador com fila de entrada — um pacote que está na fila de entrada deve esperar pela transferência através do elemento de comutação (mesmo que sua porta de saída esteja livre) porque ele está bloqueado por outro pacote na cabeça da fila. Karol [1987] demonstra que, por causa do bloqueio HOL, o comprimento da fila de entrada cresce sem limites (informalmente, isso equivale a dizer que haverá significativas perdas de pacotes) em determinadas circunstâncias assim que a taxa de chegada de pacotes no enlace de entrada alcançar apenas 58% de sua capacidade. Uma série de soluções para o bloqueio HOL é discutida por McKeown [1997b].

4.3.5 O plano de controle de roteamento

Em nossa discussão até aqui e na Figura 4.6, consideramos de modo implícito que o plano de controle de roteamento reside totalmente e é executado em um processador de roteamento dentro do roteador. O plano de controle de roteamento em âmbito de rede é, portato, descentralizado — com diferentes partes (por exemplo, de um algoritmo de roteamento) executando em diferentes roteadores e interagindo pelo envio de mensagens de controle entre si. Na verdade, os roteadores atuais na Internet e os algoritmos de

FIGURA 4.11 BLOQUEIO DE CABEÇA DE FILA EM UM COMUTADOR COM FILA DE ENTRADA

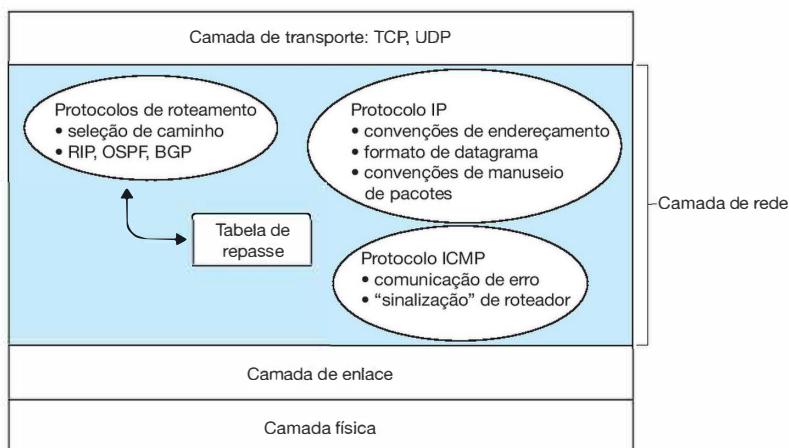
Disputa pela porta de saída no tempo *t* — um pacote escuro pode ser transferido

Elemento de comutação

Pacote claro do último retângulo sofre bloqueio HOL

Elemento de comutação

Legenda:
destinado à porta de saída mais alta
destinado à porta de saída do meio
destinado à porta de saída mais baixa

roteamento que estudaremos na Seção 4.6 operam exatamente dessa maneira. Além disso, fornecedores de roteador e comutador agrupam seu plano de dados do hardware e plano de controle do software em plataformas fechadas (porém, interoperáveis) em um produto verticalmente integrado.

Há pouco, diversos pesquisadores [Caesar, 2005a; Casado, 2009; McKeown, 2008] começaram a explorar novas arquiteturas de plano de controle de roteador, nas quais parte das funções do plano é executada nos roteadores (por exemplo, medição/relatório local do estado do enlace, instalação da tabela de repasse e manutenção), junto com o plano de dados, e parte do plano de controle pode ser implementada externamente ao roteador (por exemplo, em um servidor centralizado, que poderia realizar cálculo de rota). Uma API bem definida determina como essas duas partes interagem e se comunicam. Esses pesquisadores argumentam que a separação do plano de controle em software do plano de dados em hardware (com um mínimo de plano de controle residente no roteador) pode simplificar o roteamento, substituindo o cálculo de roteamento distribuido pelo cálculo de roteamento centralizado, e permitir a inovação na rede, aceitando que diferentes planos de controle customizados operem por planos de dados de hardware velozes.

4.4 O PROTOCOLO DA INTERNET (IP): REPASSE E ENDEREÇAMENTO NA INTERNET

Até agora discutimos endereçamento e repasse na camada de rede, sem referências a nenhuma rede de computadores específica. Nesta seção, voltaremos nossa atenção a como são feitos o endereçamento e o repasse na Internet. Veremos que o endereçamento e o repasse na Internet são componentes importantes do Protocolo da Internet (IP). Há duas versões do protocolo IP em uso hoje. Examinaremos primeiro a mais utilizada, a versão 4, que em geral é denominada apenas IPv4 [RFC 791]. Examinaremos a versão 6 do IP [RFC 2460; RFC 4291], que foi proposta para substituir o IPv4, no final desta seção.

Mas, antes de iniciar nossa investida no IP, vamos voltar um pouco atrás e considerar os componentes que formam a camada de rede da Internet. Conforme mostra a Figura 4.12, essa camada tem três componentes mais importantes. O primeiro é o protocolo IP, que é o tópico desta seção. O segundo é o componente de roteamento, que determina o caminho que um datagrama segue desde a origem até o destino. Mencionamos anteriormente que protocolos de roteamento calculam as tabelas de repasse que são usadas para transmitir pacotes pela rede. Estudaremos os protocolos de roteamento da Internet na Seção 4.6. O componente final da camada de rede é um dispositivo para comunicação de erros em datagramas e para atender requisições de certas informações de camada de rede. Examinaremos o protocolo de comunicação de erro e de informações da Internet, ICMP (Internet Control Message Protocol), na Seção 4.4.3.

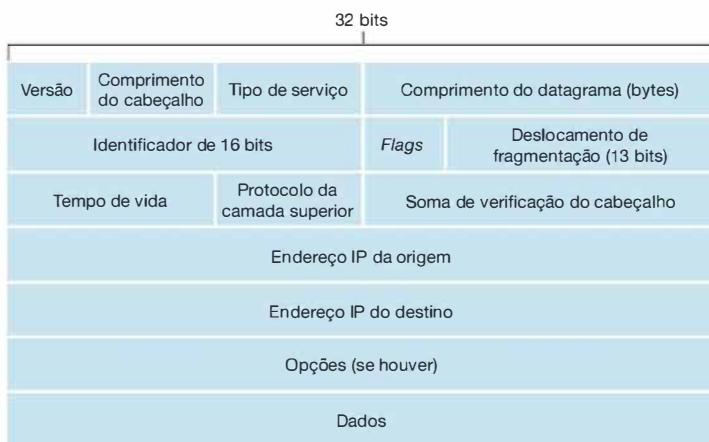



FIGURA 4.12 CONTEMPLANDO O INTERIOR DA CAMADA DE REDE DA INTERNET

4.4.1 Formato de datagrama

Lembre-se de que um pacote de camada de rede é denominado um *datagrama*. Iniciamos nosso estudo do IP com uma visão geral da sintaxe e da semântica do datagrama IPv4. Você talvez esteja pensando que nada poderia ser mais desinteressante do que a sintaxe e a semântica dos bits de um pacote. Mesmo assim, o datagrama desempenha um papel central na Internet — todos os estudantes e profissionais de rede precisam vê-lo, absorvê-lo e dominá-lo. O formato do datagrama IPv4 é mostrado na Figura 4.13. Seus principais campos são os seguintes:

- *Número da versão*. Esses quatro bits especificam a versão do protocolo IP do datagrama. Examinando o número da versão, o roteador pode determinar como interpretar o restante do datagrama IP. Diferentes versões de IP usam diferentes formatos de datagramas. O formato para a versão atual do IP (IPv4) é mostrado na Figura 4.13. O formato do datagrama para a nova versão do IP (IPv6) será discutido no final desta seção.
- *Comprimento do cabeçalho*. Como um datagrama IPv4 pode conter um número variável de opções (incluídas no cabeçalho do datagrama IPv4), esses quatro bits são necessários para determinar onde, no datagrama IP, os dados começam de fato. A maior parte dos datagramas IP não contém opções; portanto, o datagrama IP típico tem um cabeçalho de 20 bytes.
- *Tipo de serviço*. Os bits de tipo de serviço (*type of service* — TOS) foram incluídos no cabeçalho do IPv4 para poder diferenciar os diferentes tipos de datagramas IP (por exemplo, que requerem, particularmente, baixo atraso, alta vazão ou confiabilidade) que devem ser distinguidos uns dos outros. Por exemplo,

FIGURA 4.13 FORMATO DO DATAGRAMA IPv4

32 bits

Versão	Comprimento do cabeçalho	Tipo de serviço	Comprimento do datagrama (bytes)	
Identificador de 16 bits			*Flags*	Deslocamento de fragmentação (13 bits)
Tempo de vida		Protocolo da camada superior	Soma de verificação do cabeçalho	
Endereço IP da origem				
Endereço IP do destino				
Opções (se houver)				
Dados				

poderia ser útil distinguir datagramas de tempo real (como os usados por uma aplicação de telefonia IP) de tráfego que não é de tempo real (por exemplo, FTP). O nível de serviço a ser fornecido é uma questão de política determinada pelo administrador do roteador. Examinaremos em detalhes a questão do serviço diferenciado no Capítulo 7.

- *Comprimento do datagrama*. É o comprimento total do datagrama IP (cabeçalho mais dados) medido em bytes. Uma vez que esse campo tem 16 bits de comprimento, o tamanho máximo teórico do datagrama IP é 65.535 bytes. Contudo, datagramas raramente são maiores do que 1.500 bytes.
- *Identificador, flags, deslocamento de fragmentação*. Esses três campos têm a ver com a fragmentação do IP, um tópico que em breve vamos detalhar. O interessante é que a nova versão do IP, o IPv6, não permite fragmentação em roteadores.
- *Tempo de vida*. O campo de tempo de vida (*time-to-live* — TTL) é incluído para garantir que datagramas não fiquem circulando para sempre na rede (por causa de, por exemplo, um *laço* de roteamento de longa duração). Esse campo é decrementado de uma unidade cada vez que o datagrama é processado por um roteador. Se o campo TTL chegar a 0, o datagrama deve ser descartado.
- *Protocolo*. Usado somente quando um datagrama IP chega a seu destino final. O valor do campo indica o protocolo de camada de transporte específico ao qual a porção de dados desse datagrama IP deverá ser passada. Por exemplo, um valor 6 indica que a porção de dados será passada ao TCP, enquanto um valor 17 indica que os dados serão passados ao UDP. Consulte IANA Protocol Numbers [2012] para ver uma lista de todos os valores possíveis. Note que o número do protocolo no datagrama IP tem um papel análogo ao do campo de número de porta no segmento da camada de transporte. O número do protocolo é o elo entre as camadas de rede e de transporte, ao passo que o número de porta liga as camadas de transporte e de aplicação. Veremos no Capítulo 5 que o quadro de camada de enlace também tem um campo especial que liga a camada de enlace à camada de rede.
- *Soma de verificação do cabeçalho*. A soma de verificação do cabeçalho auxilia um roteador na detecção de erros de bits em um datagrama IP recebido. É calculada tratando cada 2 bytes do cabeçalho como se fossem um número e somando esses números usando complementos aritméticos de 1. Como discutimos na Seção 3.3, o complemento de 1 dessa soma, conhecida como soma de verificação da Internet, é armazenado no campo de soma de verificação. Um roteador calculará o valor da soma de verificação para cada datagrama IP recebido e detectará uma condição de erro se o valor carregado no cabeçalho do datagrama não for igual à soma de verificação calculada. Roteadores em geral descartam datagramas quando um erro é detectado.

Note que a soma de verificação deve ser recalculada e armazenada de novo em cada roteador, pois o campo TTL e, possivelmente, também os campos de opções podem mudar. Uma discussão interessante sobre algoritmos rápidos para calcular a soma de verificação da Internet é encontrada em [RFC 1071]. Uma pergunta que sempre é feita nesse ponto é: por que o TCP/IP faz verificação de erro nas camadas de transporte e de rede? Há várias razões para essa repetição. Primeiro, note que, na camada IP, a soma de verificação é calculada só para o cabeçalho IP, enquanto no TCP/UDP a soma de verificação é calculada para todo o segmento TCP/IP. Segundo, o TCP/UDP e o IP não precisam necessariamente pertencer à mesma pilha de protocolos. O TCP pode, em princípio, rodar sobre um protocolo diferente (por exemplo, ATM) e o IP pode carregar dados que não serão passados ao TCP/UDP.

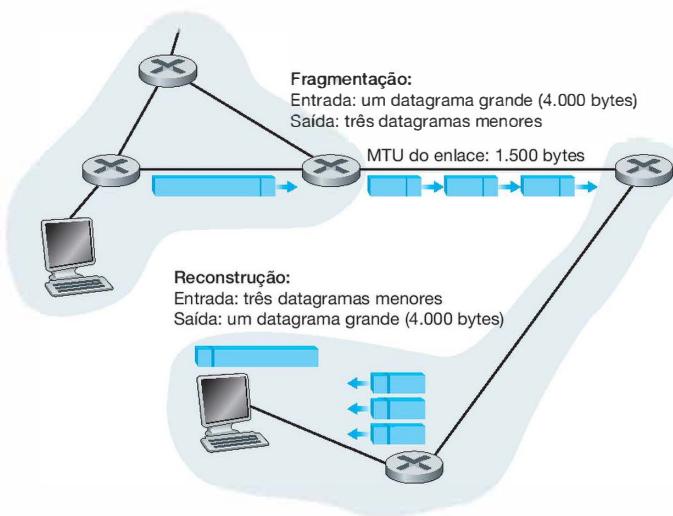
- *Endereços IP de origem e de destino*. Quando uma origem cria um datagrama, insere seu endereço IP no campo de endereço de origem IP e insere o endereço do destino final no campo de endereço de destinatário IP. Muitas vezes o hospedeiro da origem determina o endereço do destinatário por meio de uma consulta ao DNS, como discutimos no Capítulo 2. Discutiremos endereçamento IP detalhadamente na Seção 4.4.2.
- *Opções*. O campo de opções permite que um cabeçalho IP seja estendido. A intenção é que as opções de cabeçalho sejam usadas raramente — daí a decisão de poupar sobrecarga não incluindo a informação em campos de opções em todos os cabeçalhos de datagrama. Contudo, a mera existência de opções, na realidade, complica as coisas — uma vez que cabeçalhos de datagramas podem ter comprimentos variáveis, não é possível determinar *a priori* onde começará o campo de dados. Além disso, como alguns datagramas podem requerer processamento de opções e outros não, o tempo necessário para processar um datagrama IP em um roteador pode variar bastante. Essas considerações se tornam particularmente importantes para o processamento do IP em roteadores e hospedeiros de alto desempenho. Por essas e outras razões, as opções IP foram descartadas no cabeçalho da versão IPv6, como discutimos na Seção 4.4.4.
- *Dados (carga útil)*. Por fim, chegamos ao último e mais importante campo — a razão de ser do datagrama! Em muitas circunstâncias, o campo de dados do datagrama IP contém o segmento da camada de transporte (TCP ou UDP) a ser entregue ao destino. Contudo, o campo de dados pode carregar outros tipos de dados, como mensagens ICMP (discutidas na Seção 4.4.3).

Note que um datagrama IP tem um total de 20 bytes de cabeçalho (admitindo-se que não haja opções). Se o datagrama carregar um segmento TCP, então cada datagrama (não fragmentado) carrega um total de 40 bytes de cabeçalho (20 bytes de cabeçalho IP, mais 20 bytes de cabeçalho TCP) junto com a mensagem de camada de aplicação.

Fragmentação do datagrama IP

Veremos no Capítulo 5 que nem todos os protocolos de camada de enlace podem transportar pacotes do mesmo tamanho. Alguns protocolos podem transportar datagramas grandes, ao passo que outros apenas pacotes pequenos. Por exemplo, quadros Ethernet não podem conter mais do que 1.500 bytes de dados, enquanto quadros para alguns enlaces de longa distância não podem conter mais do que 576 bytes. A quantidade máxima de dados que um quadro de camada de enlace pode carregar é denominada unidade máxima de transmissão (*maximum transmission unit* — MTU). Como cada datagrama IP é encapsulado dentro do quadro de camada de enlace para ser transportado de um roteador até o roteador seguinte, a MTU do protocolo de camada de enlace estabelece um limite estrito para o comprimento de um datagrama IP. Ter uma limitação estrita para o tamanho de um datagrama IP não é grande problema. O problema é que cada um dos enlaces da rota entre remetente e destinatário pode usar diferentes protocolos de camada de enlace, e cada um desses protocolos pode ter diferentes MTUs.

Para entender melhor a questão do repasse, imagine que *você* é um roteador que está interligando diversos enlaces, cada um rodando diferentes protocolos de camada de enlace com diferentes MTUs. Suponha que você receba um datagrama IP de um enlace, verifique sua tabela de repasse para determinar o enlace de saída e perceba

que este tem uma MTU menor do que o comprimento do datagrama IP. É hora de entrar em pânico — como você vai comprimir esse datagrama IP de tamanho excessivo no campo de carga útil do quadro da camada de enlace? A solução para esse problema é fragmentar os dados do datagrama IP em dois ou mais datagramas IP menores, encapsular cada um em um quadro separado na camada de enlace e, então, enviá-los pelo enlace de saída. Cada um desses datagramas menores é denominado um **fragmento**.

Fragmentos precisam ser reconstruídos antes que cheguem à camada de transporte no destino. Na verdade, tanto o TCP quanto o UDP esperam receber da camada de rede segmentos completos, não fragmentados. Os projetistas do IPv4 perceberam que a reconstrução de datagramas nos roteadores introduziria uma complicação significativa no protocolo e colocaria um freio no desempenho do roteador. (Se você fosse um roteador, ia querer reconstruir fragmentos além de tudo mais que tem de fazer?) Seguindo o princípio de manter a simplicidade do núcleo da rede, os projetistas do IPv4 decidiram alocar a tarefa de reconstrução de datagramas aos sistemas finais, e não aos roteadores da rede.

Quando um hospedeiro destinatário recebe uma série de datagramas da mesma origem, ele precisa determinar se alguns deles são fragmentos de um datagrama original de maior tamanho. Se alguns forem fragmentos, o hospedeiro ainda deverá determinar quando recebeu o último fragmento e como os fragmentos recebidos devem ser reconstruídos para voltar à forma do datagrama original. Para permitir que o hospedeiro destinatário realize essas tarefas de reconstrução, os projetistas do IP (versão 4) criaram campos de *identificação*, *flag* e *deslocamento de fragmentação* no cabeçalho do datagrama IP. Quando um datagrama é criado, o hospedeiro remetente marca o datagrama com um número de identificação, bem como com os endereços de origem e de destino. Em geral, o hospedeiro remetente incrementa o número de identificação para cada datagrama que envia. Quando um roteador precisa fragmentar um datagrama, cada datagrama resultante (isto é, cada fragmento) é marcado com o endereço de origem, o endereço do destino e o número de identificação do datagrama original. Quando o destinatário recebe uma série de datagramas do mesmo hospedeiro remetente, pode examinar os números de identificação para determinar quais deles são, na verdade, fragmentos de um mesmo datagrama de tamanho maior. Como o IP é um serviço não confiável, é possível que um ou mais fragmentos jamais cheguem ao destino. Por essa razão, para que o hospedeiro de destino fique absolutamente seguro de que recebeu o último fragmento do datagrama original, o último datagrama tem um bit de *flag* ajustado para 0, ao passo que todos os outros têm um bit de *flag* ajustado para 1. Além disso, para que o hospedeiro destinatário determine se falta algum fragmento (e possa reconstruí-los na ordem correta), o campo de deslocamento é usado para especificar a localização exata do fragmento no datagrama IP original.

A Figura 4.14 apresenta um exemplo. Um datagrama de 4 mil bytes (20 bytes de cabeçalho IP, mais 3.980 bytes de carga útil IP) chega a um roteador e deve ser repassado a um enlace com MTU de 1.500 bytes. Isso implica que os 3.980 bytes de dados do datagrama original devem ser alocados a três fragmentos separados (cada qual é também um datagrama IP). Suponha que o datagrama original esteja marcado com o número de identificação 777. As características dos três fragmentos são mostradas na Tabela 4.2. Os valores da tabela refletem a exigência de que a quantidade de dados da carga útil original em todos os fragmentos, exceto o último, seja um múltiplo de 8 bytes, e que o valor de deslocamento seja especificado em unidades de porções de 8 bytes.

No destino, a carga útil do datagrama é passada para a camada de transporte apenas após a camada IP ter reconstruído totalmente o datagrama IP original. Se um ou mais fragmentos não chegarem ao destino, o datagrama incompleto será descartado e não será passado à camada de transporte. Mas, como aprendemos no capítulo anterior, se o TCP estiver sendo usado na camada de transporte, ele recuperará essa perda fazendo a origem retransmitir os dados do datagrama original.

Vimos que a fragmentação do IP tem papel importante na união das diversas tecnologias distintas da camada de enlace. Mas a fragmentação também tem seu preço. Primeiro, ela dificulta roteadores e sistemas finais, que precisam ser projetados para acomodar a fragmentação do datagrama e o reagrupamento. Segundo, a fragmentação pode ser usada para criar ataques DoS fatais, em que o atacante envia uma série de fragmentos estranhos e inesperados. Um exemplo clássico é o ataque Jolt2, no qual o atacante envia uma cadeia de pequenos fragmentos para o computador-alvo, nenhum dos quais com um deslocamento de zero. O alvo pode se recolher ao tentar

FIGURA 4.14 FRAGMENTAÇÃO E RECONSTRUÇÃO IP

reconstruir datagramas pelos pacotes degenerados. Outra classe de *exploits* envia sobreposição de fragmentos IP, ou seja, fragmentos cujos valores de deslocamento são definidos para que não se alinhem corretamente. Sistemas operacionais vulneráveis, sem saber o que fazer com a sobreposição de fragmentos, podem pifar [Skoudis, 2006]. Como veremos no final desta seção, uma nova versão do protocolo IP, o IPv6, põe fim por completo à fragmentação, o processamento do pacote IP e tornando o IP menos vulnerável a ataques.

No site de apoio deste livro apresentamos um applets Java que gera fragmentos. Basta informar o tamanho do datagrama que está chegando, a MTU e a identificação do datagrama, e o applets gera automaticamente os fragmentos para você.

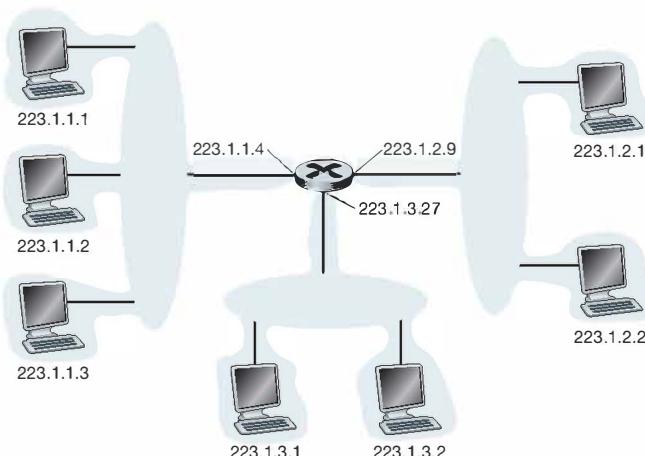
TABELA 4.2 FRAGMENTOS IP

Fragmento	Bytes	ID	Deslocamento	*Flag*
1º fragmento	1.480 bytes no campo de dados do datagrama IP	identificação = 777	0 (o que significa que os dados devem ser inseridos a partir do byte 0)	1 (o que significa que há mais)
2º fragmento	1.480 bytes de dados	identificação = 777	185 (o que significa que os dados devem ser inseridos a partir do byte 1.480. Note que 185 x 8 = 1.480)	1 (o que significa que há mais)
3º fragmento	1.020 bytes de dados (= 3.980 –1.480 –1.480)	identificação = 777	370 (o que significa que os dados devem ser inseridos a partir do byte 2.960. Note que 370 x 8 = 2.960)	0 (o que significa que esse é o último fragmento)

4.4.2 Endereçamento IPv4

Agora voltaremos nossa atenção ao endereçamento IPv4. Embora você talvez esteja pensando que o endereçamento seja um tópico complicado, esperamos que, no final deste capítulo, convença-se de que o endereçamento na Internet não é apenas um tópico rico, cheio de sutilezas e interessante, mas também de crucial importância. Tratamentos excelentes do endereçamento IPv4 podem ser encontrados em 3Com Addressing [2012] e no primeiro capítulo de Stewart [1999].

Antes de discutirmos o endereçamento IP, contudo, temos de falar um pouco sobre como hospedeiros e roteadores estão interconectados na rede. Um hospedeiro em geral tem apenas um único enlace com a rede; quando o IP no hospedeiro quer enviar um datagrama, ele o faz por meio desse enlace. A fronteira entre o hospedeiro e o enlace físico é denominada **interface**. Agora considere um roteador e suas interfaces. Como a tarefa de um roteador é receber um datagrama em um enlace e repassá-lo a algum outro enlace, ele necessariamente estará ligado a dois ou mais enlaces. A fronteira entre o roteador e qualquer um desses enlaces também é denominada uma interface. Assim, um roteador tem múltiplas interfaces, uma para cada enlace. Como todos os hospedeiros e roteadores podem enviar e receber datagramas IP, o IP exige que cada interface tenha seu próprio endereço IP. Desse modo, um endereço IP está tecnicamente associado com uma interface, e não com um hospedeiro ou um roteador que contém aquela interface.

Cada endereço IP tem comprimento de 32 bits (equivalente a 4 bytes). Portanto, há um total de 2^{32} endereços IP possíveis. Fazendo uma aproximação de 2^{10} por 10^3, é fácil ver que há cerca de 4 bilhões de endereços IP possíveis. Esses endereços são escritos em **notação decimal separada por pontos** (*dotted-decimal notation*), na qual cada byte do endereço é escrito em sua forma decimal e separado dos outros bytes do endereço por um ponto. Por exemplo, considere o endereço IP 193.32.216.9. O 193 é o número decimal equivalente aos primeiros 8 bits do endereço; o 32 é o decimal equivalente ao segundo conjunto de 8 bits do endereço e assim por diante. Por conseguinte, o endereço 193.32.216.9, em notação binária, é:

11000001 00100000 11011000 00001001

Cada interface em cada hospedeiro e roteador da Internet global tem de ter um endereço IP globalmente exclusivo (exceto as interfaces por trás de NATs, como discutiremos no final desta seção). Contudo, os endereços não podem ser escolhidos de qualquer maneira. Uma parte do endereço IP de uma interface será determinada pela sub-rede à qual ela está conectada.

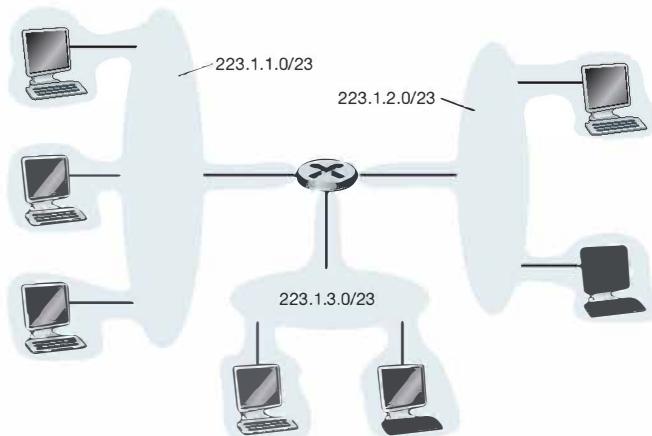
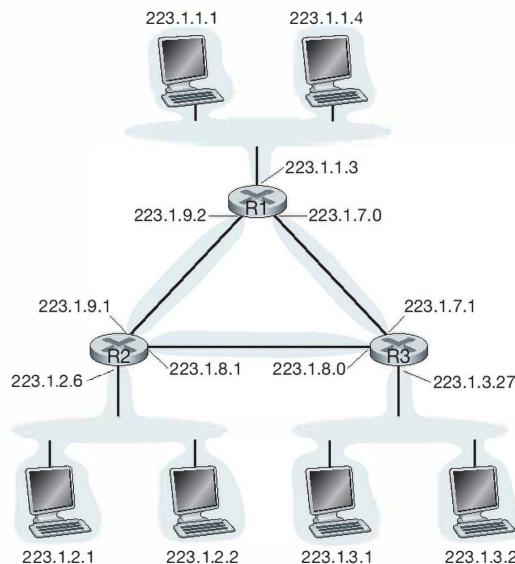
A Figura 4.15 fornece um exemplo de endereçamento IP e interfaces. Nela, um roteador (com três interfaces) é usado para interconectar sete hospedeiros. Examine os endereços IP atribuídos às interfaces de hospedeiros e roteadores; há diversas particularidades a notar. Todos os três hospedeiros da parte superior esquerda da Figura 4.15 — e também a interface do roteador ao qual estão ligados — têm um endereço IP na forma 223.1.1.xxx, ou seja, todos têm os mesmos 24 bits mais à esquerda em seus endereços IP. As quatro interfaces também estão interconectadas por uma rede *que não contém nenhum roteador*. Essa rede poderia ser interconectada por uma LAN Ethernet, caso em que as interfaces seriam conectadas por um comutador Ethernet (conforme discutiremos no Capítulo 5) ou por um ponto de acesso sem fio (como veremos no Capítulo 6). Vamos representar essa rede sem roteador que conecta esses hospedeiros como uma nuvem por enquanto, mas entraremos nos detalhes internos dessas redes nos Capítulos 5 e 6.

Na terminologia IP, essa rede que interconecta três interfaces de hospedeiros e uma interface de roteador forma uma **sub-rede** [RFC 950]. (Na literatura da Internet, uma sub-rede também é denominada uma *rede IP* ou simplesmente uma *rede*.) O endereçamento IP designa um endereço a essa sub-rede: 223.1.1.0/24, no qual a notação /24, às vezes conhecida como uma **máscara de sub-rede**, indica que os 24 bits mais à esquerda do conjunto de 32 bits definem o endereço da sub-rede. Assim, a sub-rede 223.1.1.0/24 consiste em três interfaces de hospedeiros (223.1.1.1, 223.1.1.2 e 223.1.1.3) e uma interface de roteador (223.1.1.4). Quaisquer hospedeiros adicionais ligados à sub-rede 223.1.1.0/24 seriam obrigados a ter um endereço na forma 223.1.1.xxx. Há duas sub-redes adicionais mostradas na Figura 4.15: a sub-rede 223.1.2.0/24 e a sub-rede 223.1.3.0/24. A Figura 4.16 ilustra as três sub-redes IP presentes na Figura 4.15.

A definição IP de uma sub-rede não está restrita a segmentos Ethernet que conectam múltiplos hospedeiros a uma interface de roteador. Para entender melhor, considere a Figura 4.17, que mostra três roteadores interconectados por enlaces ponto a ponto. Cada roteador tem três interfaces, uma para cada enlace ponto a ponto e uma para o enlace para um grupo, que conecta diretamente o roteador a um par de hospedeiros. Que sub-redes IP estão presentes aqui? Três sub-redes, 223.1.1.0/24, 223.1.2.0/24 e 223.1.3.0/24, semelhantes às que encontramos na Figura 4.15. Mas note que também há três sub-redes adicionais nesse exemplo: uma sub-rede, 223.1.9.0/24, para as interfaces que conectam os roteadores R1 e R2; outra, 223.1.8.0/24, para as interfaces que conectam os roteadores

FIGURA 4.15 ENDEREÇOS DE INTERFACES E SUB-REDES

R2 e R3, e uma terceira, 223.1.7.0/24, para as interfaces que conectam os roteadores R3 e R1. Para um sistema geral interconectado de roteadores e hospedeiros, podemos usar a seguinte receita para definir as sub-redes no sistema:

> *Para determinar as sub-redes, destaque cada interface de seu hospedeiro ou roteador, criando ilhas de redes isoladas com interfaces fechando as terminações das redes isoladas. Cada uma dessas redes isoladas é denominada* ***sub-rede****.*

Se aplicarmos esse procedimento ao sistema interconectado da Figura 4.17, teremos seis ilhas ou sub-redes.

Fica claro, com essa discussão, que uma organização (tal como uma empresa ou instituição acadêmica) que tenha vários segmentos Ethernet e enlaces ponto a ponto terá várias sub-redes, e todos os equipamentos e dispositivos em uma dada sub-rede terão o mesmo endereço de sub--rede. Em princípio, as diversas sub-redes poderiam ter endereços bem diferentes. Entretanto, na prática, os endereços de sub-rede com frequência têm muito em comum. Para entender por que, voltemos nossa atenção ao modo como o endereçamento é manipulado na Internet global.

A estratégia de atribuição de endereços da Internet é conhecida como **roteamento interdomínio sem classes** (*Classless Interdomain Routing* — CIDR), que se pronuncia "sáider", como a palavra *cider* (cidra) em inglês [RFC 4632]. O CIDR generaliza a noção de endereçamento de sub-rede. Como acontece com o endereçamento de sub-redes, o endereço IP de 32 bits é dividido em duas partes e, mais uma vez, tem a forma decimal com pontos de separação *a.b.c.d/x*, em que *x* indica o número de bits da primeira parte do endereço.

Os *x* bits mais significativos de um endereço na forma *a.b.c.d/x* constituem a parcela da rede do endereço IP e costumam ser denominados **prefixo** (ou *prefixo de rede*). Uma organização em geral recebe um bloco de endereços contíguos, isto é, uma faixa de endereços com um prefixo comum (ver quadro "Princípios na prática"). Nesse caso, os endereços IP de equipamentos e dispositivos dentro da organização compartilharão o prefixo comum. Quando estudarmos, na Seção 4.6, o protocolo de roteamento BGP da Internet, veremos que somente esses *x* bits indicativos do prefixo são considerados por roteadores que estão fora da rede da organização. Isto é, quando um roteador de fora repassar um datagrama cujo endereço de destino esteja dentro da organização, terá de considerar apenas os *x* bits indicativos do endereço. Isso reduz de modo considerável o tamanho da tabela de repasse nesses roteadores, visto que um *único* registro da forma *a.b.c.d/x* será suficiente para transmitir pacotes para *qualquer* destino dentro da organização.

Os restantes (32 – *x*) bits de um endereço podem ser considerados os bits que distinguem os equipamentos e dispositivos *dentro* da organização e todos eles têm o mesmo prefixo de rede. Eles serão os bits considerados no repasse de pacotes em roteadores *dentro* da organização. Esses bits de ordem mais baixa podem (ou não) ter uma

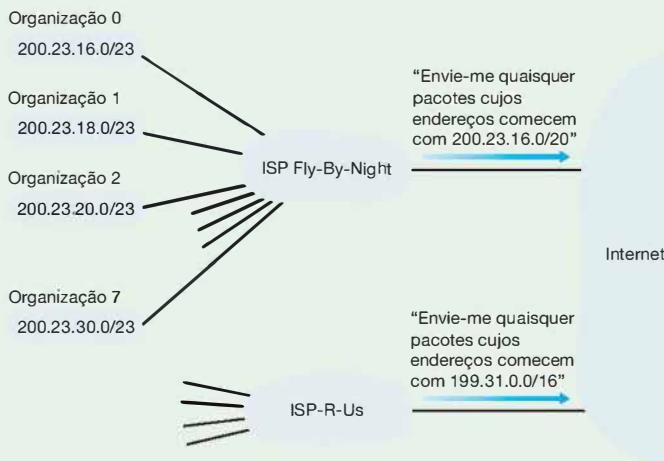
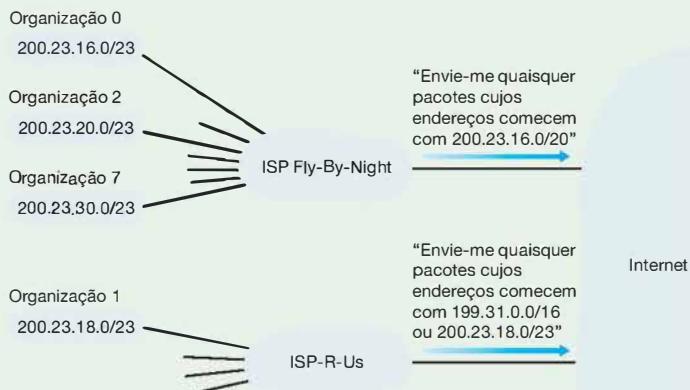
FIGURA 4.16 ENDEREÇOS DE SUB-REDES

FIGURA 4.17 TRÊS ROTEADORES INTERCONECTANDO SEIS SUB-REDES

estrutura adicional de sub-rede tal como a discutida anteriormente. Por exemplo, suponha que os primeiros 21 bits do endereço a.b.c.d/21, por assim dizer, "ciderizado", especificam o prefixo da rede da organização e são comuns aos endereços IP de todos os hospedeiros da organização. Os 11 bits restantes então identificam os hospedeiros específicos da organização. A estrutura interna da organização poderia ser tal que esses 11 bits mais à direita seriam usados para criar uma sub-rede dentro da organização, como discutido antes. Por exemplo, a.b.c.d/24 poderia se referir a uma sub-rede específica dentro da organização.

Antes da adoção do CIDR, os tamanhos das parcelas de um endereço IP estavam limitados a 8, 16 ou 24 bits, um esquema de endereçamento conhecido como **endereçamento com classes**, já que sub-redes com endereços

de sub-rede de 8, 16 e 24 eram conhecidas como redes de classe A, B e C, respectivamente. A exigência de que a parcela da sub-rede de um endereço IP tenha exatos 1, 2 ou 3 bytes há muito tempo se mostrou problemática para suportar o rápido crescimento do número de organizações com sub-redes de pequeno e médio portes. Uma sub-rede de classe C (/24) poderia acomodar apenas $2^8 - 2 = 254$ hospedeiros (dois dos $2^8 = 256$ endereços são reservados para uso especial) — muito pequena para inúmeras organizações. Contudo, uma sub-rede de classe B (/16), que suporta até 65.634 hospedeiros, seria grande demais. Com o endereçamento com classes, uma organização com, digamos, dois mil hospedeiros, recebia um endereço de sub-rede de classe B (/16), o que resultava no rápido esgotamento do espaço de endereços de classe B e na má utilização do espaço de endereço alocado. Por exemplo, uma organização que usasse um endereço de classe B para seus dois mil hospedeiros, recebia espaço de endereços suficiente para até 65.534 interfaces — deixando mais de 63 mil endereços sem uso e que não poderiam ser utilizados por outras organizações.

Seríamos omissos se não mencionássemos ainda outro tipo de endereço IP, o endereço de difusão 255.255.255.255. Quando um hospedeiro emite um datagrama com endereço de destino 255.255.255.255, a mensagem é entregue a todos os hospedeiros na mesma sub-rede. Os roteadores também têm a opção de repassar a mensagem para suas sub-redes vizinhas (embora em geral não o façam).

Agora que já estudamos o endereçamento IP detalhadamente, precisamos saber, antes de qualquer coisa, como hospedeiros e sub-redes obtêm seus endereços. Vamos começar examinando como uma organização obtém um bloco de endereços para seus equipamentos e, então, veremos como um equipamento (tal como um hospedeiro) recebe um endereço do bloco de endereços da organização.

PRINCÍPIOS NA PRÁTICA

Esse exemplo de um ISP que conecta oito organizações com a Internet também ilustra de maneira elegante como endereços "ciderizados" alocados com cuidado facilitam o roteamento. Suponha, como mostra a Figura 4.18, que o ISP (que chamaremos de ISP Fly-By-Night) anuncie ao mundo exterior que devem ser enviados a ele quaisquer datagramas cujos primeiros 20 bits de endereço sejam iguais a 200.23.16.0/20. O resto do mundo não precisa saber que dentro do bloco de endereços 200.23.16.0/20 existem, na verdade, oito outras organizações, cada qual com suas próprias sub-redes. A capacidade de usar um único prefixo de rede para anunciar várias redes costuma ser denominada **agregação de endereços** (e também **agregação de rotas** ou **resumo de rotas**).

A agregação de endereços funciona muito bem quando estes são alocados em blocos ao ISP e, então, pelo ISP às organizações clientes. Mas o que ocorre quando os endereços não são alocados dessa maneira hierárquica? O que aconteceria, por exemplo, se o ISP Fly-By-Night adquirisse o ISP-R-Us e então ligasse a Organização 1 com a Internet por meio de sua subsidiária ISP-R-Us? Como mostra a Figura 4.18, a subsidiária ISP-R-Us é dona do bloco de endereços 199.31.0.0/16, mas os endereços IP da Organização 1 infelizmente estão fora desse bloco. O que deveria ser feito nesse caso? Com certeza, a Organização 1 poderia renumerar todos os seus roteadores e hospedeiros para que seus endereços ficassem dentro do bloco de endereços do ISP-R-Us. Mas essa é uma solução dispendiosa, e a Organização 1 poderia muito bem preferir mudar para outra subsidiária no futuro. A solução em geral adotada é a Organização 1 manter seus endereços IP em 200.23.18.0/23. Nesse caso, como mostra a Figura 4.19, o ISP Fly-By-Night continua a anunciar o bloco de endereços 200.23.16.0/20 e o ISP-R-Us continua a anunciar 199.31.0.0/16. Contudo, o ISP-R-Us agora anuncia *também* o bloco de endereços para a Organização 1, 200.23.18.0/23. Quando outros roteadores da Internet virem os blocos de endereços 200.23.16.0/20 (do ISP Fly-By-Night) e 200.23.18.0/23 (do ISP-R-Us) e quiserem rotear para um endereço no bloco 200.23.18.0/23, usarão a regra de correspondência com o prefixo mais longo (veja Seção 4.2.2) e rotearão para o ISP-R-Us, já que ele anuncia o prefixo de endereço mais longo (mais específico) que combina com o endereço de destino.

FIGURA 4.18 ENDEREÇAMENTO HIERÁRQUICO E AGREGAÇÃO DE ROTAS

FIGURA 4.19 ISP-R-Us TEM UMA ROTA MAIS ESPECÍFICA PARA A ORGANIZAÇÃO 1

Obtenção de um bloco de endereços

Para obter um bloco de endereços IP para utilizar dentro da sub-rede de uma organização, um administrador de rede poderia, primeiro, contatar seu ISP, que forneceria endereços a partir de um bloco maior de endereços que já estão alocados ao ISP. Por exemplo, o próprio ISP pode ter recebido o bloco de endereços 200.23.16.0/20. O ISP, por sua vez, dividiria seu bloco de endereços em oito blocos de endereços contíguos, do mesmo tamanho,

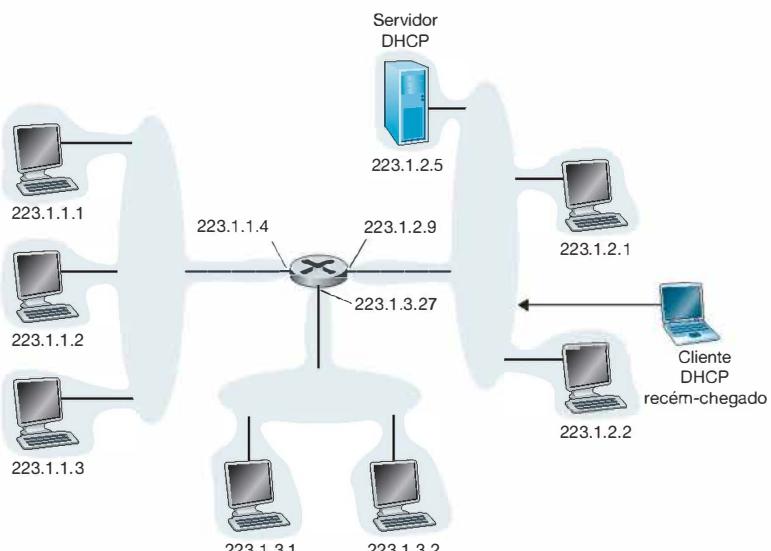
e daria um deles a cada uma de um conjunto de oito organizações suportadas por ele, como demonstrado a seguir. (Sublinhamos a parte da sub-rede desses endereços para melhor visualização.)

Bloco do ISP	200.23.16.0/20	11001000	00010111	00010000	00000000
Organização 0	200.23.16.0/23	11001000	00010111	00010000	00000000
Organização 1	200.23.18.0/23	11001000	00010111	00010010	00000000
Organização 2	200.23.20.0/23	11001000	00010111	00010100	00000000
...	...		...		
Organização 7	200.23.30.0/23	11001000	00010111	00011110	00000000

Embora a obtenção de um conjunto de endereços de um ISP seja um modo de conseguir um bloco de endereços, não é o único. Claro, também deve haver um modo de o próprio ISP obter um bloco de endereços. Há uma autoridade global que tenha a responsabilidade final de gerenciar o espaço de endereços IP e alocar blocos a ISPs e outras organizações? Sem dúvida que há! Endereços IP são administrados sob a autoridade da Internet Corporation for Assigned Names and Numbers (ICANN) [ICANN, 2012], com base em diretrizes estabelecidas no RFC 2050. O papel da ICANN, uma organização sem fins lucrativos [NTIA, 1998], não é apenas alocar endereços IP, mas também administrar os servidores DNS raiz. Essa organização também tem a controvertida tarefa de atribuir nomes de domínio e resolver disputas de nomes de domínio. A ICANN aloca endereços a serviços regionais de registro da Internet (por exemplo, ARIN, RIPE, APNIC e LACNIC), que, juntos, formam a Address Supporting Organization da ICANN [ASO-ICANN, 2012], e é responsável pela alocação/ administração de endereços dentro de suas regiões.

Obtenção de um endereço de hospedeiro: o Protocolo de Configuração Dinâmica de Hospedeiros (DHCP)

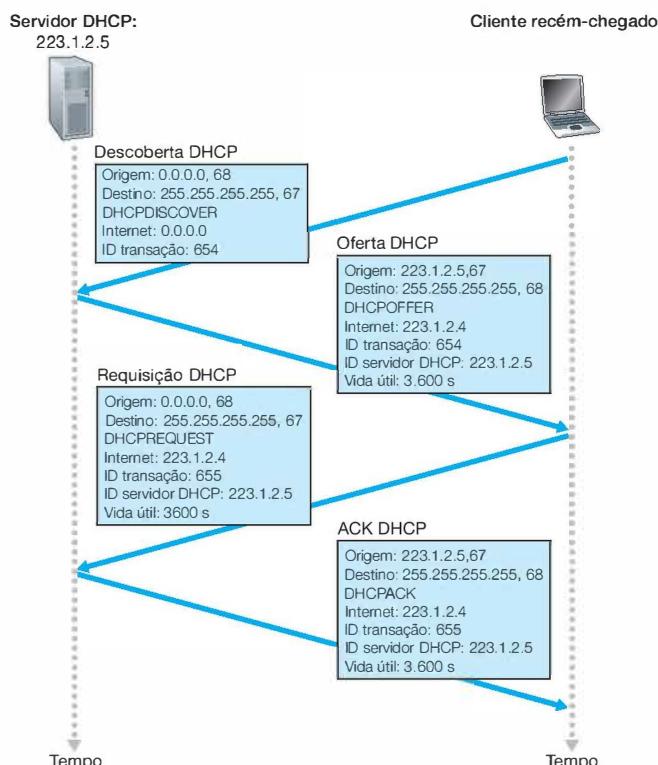
Tão logo tenha obtido um bloco de endereços, uma organização pode atribuir endereços IP individuais às interfaces de hospedeiros e roteadores. Em geral, um administrador de sistemas configurará de modo manual os endereços IP no roteador (muitas vezes remotamente, com uma ferramenta de gerenciamento de rede). Os endereços dos hospedeiros podem também ser configurados manualmente, mas essa tarefa costuma ser feita usando o **Protocolo de Configuração Dinâmica de Hospedeiros (DHCP)** [RFC 2131]. O DHCP permite que um hospedeiro obtenha (seja alocado a) um endereço IP de maneira automática. Um administrador de rede pode configurar o DHCP para que determinado hospedeiro receba o mesmo endereço IP toda vez que se conectar, ou um hospedeiro pode receber um **endereço IP temporário** diferente sempre que se conectar. Além de receber um endereço IP temporário, o DHCP também permite que o hospedeiro descubra informações adicionais, como a máscara de sub-rede, o endereço do primeiro roteador (em geral chamado de roteador de borda padrão — *default gateway*) e o endereço de seu servidor DNS local.

Por causa de sua capacidade de automatizar os aspectos relativos à rede da conexão de um hospedeiro, o DHCP é em geral denominado um **protocolo *plug and play***. Essa capacidade o torna *muito* atraente para o administrador de rede que, caso contrário, teria de executar essas tarefas manualmente! O DHCP também está conquistando ampla utilização em redes residenciais de acesso à Internet e em LANs sem fio, nas quais hospedeiros entram e saem da rede com frequência. Considere, por exemplo, um estudante que leva seu laptop do quarto para a biblioteca, para a sala de aula. É provável que ele se conecte a uma nova sub-rede em cada um desses lugares e, por conseguinte, precisará de um novo endereço IP em cada um deles. O DHCP é ideal para

essa situação, pois há muitos usuários em trânsito e os endereços são utilizados apenas por um tempo limitado. O DHCP é, de maneira semelhante, útil em redes domésticas de acesso. Como exemplo, considere um ISP residencial que tem dois mil clientes, porém, nunca mais de 400 estão on-line ao mesmo tempo. Nesse caso, em vez de precisar de um bloco de 2.048 endereços, um servidor DHCP que designa endereços dinamicamente só precisa de um bloco de 512 endereços (por exemplo, um bloco da forma a.b.c.d/23). À medida que hospedeiros entram e saem da rede, o servidor DHCP precisa atualizar sua lista de endereços IP disponíveis. Toda vez que um hospedeiro se conecta à rede, o servidor DHCP designa a ele um endereço arbitrário do seu reservatório de endereços disponíveis; toda vez que um hospedeiro sai, o endereço é devolvido ao reservatório.

O DHCP é um protocolo cliente-servidor. Em geral o cliente é um hospedeiro recém-chegado que quer obter informações sobre configuração da rede, incluindo endereço IP, para si mesmo. Em um caso mais simples, cada sub-rede (no sentido de endereçamento da Figura 4.17) terá um servidor DHCP. Se não houver um servidor na sub-rede, é necessário um agente relé DHCP (normalmente um roteador) que sabe o endereço de um servidor DHCP para tal rede. A Figura 4.20 ilustra um servidor DHCP conectado à rede 223.1.2/24, servindo o roteador de agente de repasse para clientes que chegam conectados às sub-redes 223.1.1/24 e 223.1.3/24. Em nossa discussão a seguir, admitiremos que um servidor DHCP está disponível na sub-rede.

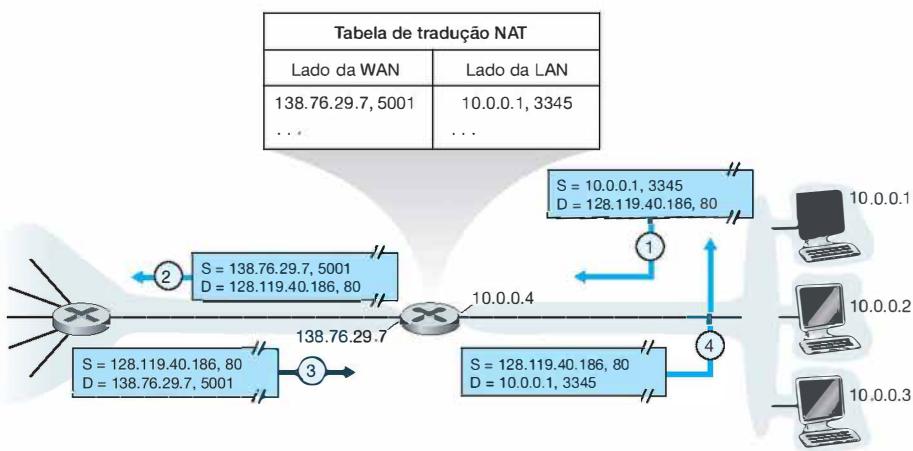
Para um hospedeiro recém-chegado, o protocolo DHCP é um processo de quatro etapas, como mostrado na Figura 4.21 para a configuração de rede mostrada na Figura 4.20. Nesta figura, "Internet" (significando "seu endereço na Internet") indica que o endereço está sendo alocado para um cliente recém-chegado. As quatro etapas são:

- *Descoberta do servidor DHCP.* A primeira tarefa de um hospedeiro recém-chegado é encontrar um servidor DHCP com quem interagir. Isso é feito utilizando uma **mensagem de descoberta DHCP**, a qual o cliente envia dentro de um pacote UDP para a porta 67. O pacote UDP é envolvido em um datagrama IP. Mas para quem esse datagrama deve ser enviado? O hospedeiro não sabe o endereço IP da rede à qual está se conectando, muito menos o endereço de um servidor DHCP para essa rede. Desse modo, o cliente DHCP cria um datagrama IP contendo sua mensagem de descoberta DHCP com o endereço IP de destino de difusão 255.255.255.255 e um endereço IP destinatário "desse hospedeiro" 0.0.0.0. O cliente DHCP

FIGURA 4.20 CENÁRIO CLIENTE-SERVIDOR DHCP

transmite o datagrama IP por difusão à camada de enlace que, então, transmite esse quadro para todos os nós conectados à sub-rede (discutiremos os detalhes sobre *difusão* da camada de enlace na Seção 5.4).

- *Oferta(s) dos servidores DHCP.* Um servidor DHCP que recebe uma mensagem de descoberta DHCP responde ao cliente com uma **mensagem de oferta DHCP**, transmitida por difusão a todos os nós presentes na sub-rede, novamente utilizando o endereço IP de transmissão 255.255.255.255. (Você poderia questionar que essa resposta do servidor também deve ser transmitida por difusão.) Como diversos servidores DHCP podem estar presentes na sub-rede, o cliente pode se dar ao luxo de escolher dentre as muitas ofertas. Cada mensagem de oferta do servidor contém o ID de transação da mensagem de descoberta recebida, o endereço IP proposto para o cliente, a máscara da rede e o **tempo de concessão do endereço IP** — o tempo pelo qual o endereço IP será válido. É comum o servidor definir o tempo de concessão para várias horas ou dias [Droms, 2002].
- *Solicitação DHCP.* O cliente recém-chegado escolherá dentre uma ou mais ofertas do servidor e responderá à sua oferta selecionada com uma **mensagem de solicitação DHCP**, repetindo os parâmetros de configuração.
- *DHCP ACK.* O servidor responde a mensagem de requisição DHCP com uma **mensagem DHCP ACK**, confirmando os parâmetros requisitados.

Uma vez que o cliente recebe o DHCP ACK, a interação é concluída e ele pode usar o endereço IP alocado pelo DHCP pelo tempo de concessão. Caso queira usar seu endereço após a expiração da concessão, o DHCP também fornece um mecanismo que permite ao cliente renovar sua concessão sobre um endereço IP.

FIGURA 4.21 INTERAÇÃO CLIENTE-SERVIDOR DHCP

O valor da capacidade *plug and play* do DHCP é clara, considerando o fato de que a alternativa é configurar manualmente um endereço IP do hospedeiro. Considere o aluno que se locomove da sala de aula para a biblioteca até seu dormitório com um laptop, entra em uma nova sub-rede, obtendo, assim, um novo endereço IP em cada local. É inimaginável que um administrador de sistema tivesse de reconfigurar laptops em cada local, e poucos alunos (exceto os que estão tendo aula de rede de computadores!) teriam condições de configurar seus laptops manualmente. Entretanto, pelo aspecto da mobilidade, o DHCP possui desvantagens. Como um novo endereço IP é obtido de um novo DHCP toda vez que um nó se conecta a uma nova sub-rede, uma conexão TCP com uma aplicação remota não pode ser mantida enquanto o nó móvel se locomove entre as sub-redes. No Capítulo 6, analisaremos o IP móvel — uma extensão recente para a infraestrutura IP que permite que um nó móvel utilize um único endereço permanente à medida que se locomove entre as sub-redes. Mais detalhes sobre o DHCP podem ser encontrados em Droms [2002] e dhc [2012]. Uma implementação de referência de código-fonte aberto do DHCP está disponível em Internet System Consortium [ISC, 2012].

Tradução de endereços na rede (NAT)

Dada a nossa discussão sobre endereços de Internet e o formato do datagrama IPv4, agora estamos bem cientes de que todo equipamento que utiliza IP precisa de um endereço IP. Isso parece significar que, com a proliferação de sub-redes de pequenos escritórios e de escritórios residenciais (*small office home office* — SOHO), sempre que um desses escritórios quiser instalar uma LAN para conectar várias máquinas, o ISP precisará alocar uma faixa de endereços para atender a todas as máquinas que usam IP ali. Se a sub-rede ficasse maior (por exemplo, as crianças da casa não só têm seus próprios computadores, mas também smartphones e jogos Game Boys em rede), seria preciso alocar um bloco de endereços maior. Mas, e se o ISP já tiver alocado as porções contíguas da faixa de endereços utilizada atualmente por esse escritório residencial? E, antes de tudo, qual é o proprietário típico de uma residência que quer (ou precisaria) saber como administrar endereços IP? Felizmente, há uma abordagem mais simples da alocação de endereços que vem conquistando uma utilização crescente nesses tipos de cenários: a **tradução de endereços de rede** (*network address translation* — **NAT**) [RFC 2663; RFC 3022; Zhang, 2007].

A Figura 4.22 mostra a operação de um roteador que utiliza NAT. Esse roteador, que fica na residência, tem uma interface que faz parte da rede residencial do lado direito da Figura 4.22. O endereçamento dentro dessa rede é exatamente como vimos antes — todas as quatro interfaces da rede têm o mesmo endereço de sub-rede, 10.00.0/24. O espaço de endereço 10.0.0.0/8 é uma das três porções do espaço de endereço IP reservado pelo [RFC 1918] para uma rede privada, ou um **domínio** com endereços privados, tal como a rede residencial da Figura 4.22. *Um domínio com endereços privados* refere-se a uma rede cujos endereços somente têm significado para equipamentos pertencentes àquela rede. Para ver por que isso é importante, considere o fato de haver centenas de milhares de redes residenciais, muitas delas utilizando o mesmo espaço de endereço 10.0.0.0/24. Equipamentos que pertencem a determinada rede residencial podem enviar pacotes entre si utilizando o endereçamento 10.0.0.0/24. Contudo, é claro que pacotes repassados da rede residencial para a Internet global não podem usar esses endereços (nem como de origem, nem como de destino) porque há centenas de milhares de redes utilizando esse bloco de endereços. Isto é, os endereços 10.0.0.0/24 somente podem ter significado dentro daquela rede residencial. Mas, se endereços privados têm significado apenas dentro de uma dada rede, como o endereçamento é administrado quando pacotes são recebidos ou enviados para a Internet global, onde os endereços são necessariamente exclusivos? A resposta será dada pelo estudo da NAT.

O roteador que usa NAT *não parece* um roteador para o mundo externo, pois se comporta como um equipamento *único* com um *único* endereço IP. Na Figura 4.22, todo o tráfego que sai do roteador residencial para a Internet tem um endereço IP de origem 138.76.29.7, e todo o tráfego que entra nessa rede tem de ter endereço de destino 138.76.29.7. Na essência, o roteador que usa NAT está ocultando do mundo exterior os detalhes da rede residencial. (A propósito, você talvez esteja imaginando onde os computadores de redes residenciais obtêm seus endereços e onde o roteador obtém seu endereço IP exclusivo. A resposta é quase sempre a mesma — DHCP! O roteador obtém seu endereço do servidor DHCP do ISP e roda um servidor DHCP para fornecer endereços a computadores que estão no espaço de endereços NAT da rede residencial controlado pelo DHCP.)

FIGURA 4.22 TRADUÇÃO DE ENDEREÇOS DE REDE (S = ORIGEM, D = DESTINO)

Tabela de tradução NAT	
Lado da WAN	Lado da LAN
138.76.29.7, 5001	10.0.0.1, 3345
. . .	. . .

Se todos os datagramas que chegam ao roteador NAT provenientes da WAN tiverem o mesmo endereço IP de destino (especificamente, o endereço da interface do roteador NAT do lado da WAN), então como o roteador sabe para qual hospedeiro interno deve repassar um datagrama? O truque é utilizar uma **tabela de tradução NAT** no roteador NAT e incluir nos registros da tabela números de portas, bem como endereços IP.

Considere o exemplo da Figura 4.22. Suponha que um usuário que está utilizando o hospedeiro 10.0.0.1 da rede residencial requisite uma página de algum servidor Web (porta 80) cujo endereço IP é 128.119.40.186. O hospedeiro 10.0.0.1 escolhe o número de porta de origem (arbitrário) 3345 e envia o datagrama para dentro da LAN. O roteador NAT recebe o datagrama, gera um novo número de porta de origem, 5001, para o datagrama, substitui o endereço IP de origem por seu endereço IP do lado da WAN, 138.76.29.7, e substitui o número de porta de origem original, 3345, pelo novo número de porta de origem, 5001. Ao gerar um novo número de porta de origem, o roteador NAT pode selecionar qualquer número de porta de origem que não esteja correntemente na tabela de tradução NAT. (Note que, como o comprimento de um campo de número de porta é 16 bits, o protocolo NAT pode suportar mais de 60 mil conexões simultâneas com um único endereço IP do lado da WAN para o roteador!) A NAT no roteador também adiciona um registro à sua tabela de tradução NAT. O servidor Web, totalmente alheio ao fato de que o datagrama que está chegando com uma requisição HTTP foi manipulado pelo roteador NAT, responde com um datagrama cujo endereço de destino é o endereço IP do roteador NAT, e cujo número de porta de destino é 5001. Quando esse datagrama chega ao roteador NAT, ele indexa a tabela de tradução NAT usando o endereço IP de destino e o número de porta de destino para obter o endereço IP (10.0.0.1) e o número de porta de destino (3345) adequados para o navegador na rede residencial. O roteador então reescreve o endereço de destino e o número de porta de destino do datagrama e o repassa para a rede residencial.

A NAT conquistou ampla aceitação nos últimos anos. Mas devemos mencionar que muitos puristas da comunidade da IETF têm grandes restrições à NAT. Primeiro, argumentam, a finalidade dos números de portas é endereçar processos, e não hospedeiros. (De fato, a violação dessa regra pode causar problemas para servidores que rodam em redes residenciais, pois, como vimos no Capítulo 2, processos servidores esperam pela chegada de requisições em números de portas bem conhecidos.) Segundo, alegam que roteadores devem processar pacotes apenas até a camada 3. Terceiro, discutem, o protocolo NAT viola o argumento denominado fim a fim; isto é, hospedeiros devem falar diretamente uns com os outros, sem a interferência de nós que modifiquem endereços IP e números de portas. Quarto, argumentam que deveríamos usar o IPv6 (veja Seção 4.4.4) para resolver a escassez de endereços IP, e não tentar resolver o problema imprudentemente com uma solução temporária como a NAT. Mas, gostemos ou não, a NAT tornou-se um componente importante da Internet.

Outro problema importante da NAT é que ela interfere com aplicações P2P, incluindo as de compartilhamento de arquivos P2P e as de voz sobre IP. Lembre-se de que, no Capítulo 2, dissemos que, em uma aplicação P2P, qualquer par A participante deve poder iniciar uma conexão TCP com qualquer outro par B participante. A essência do problema é que, se o par B estiver por trás de uma NAT, não poderá agir como um servidor e aceitar conexões TCP. Como veremos nos problemas de fixação, essa questão da NAT pode ser contornada se o par A se conectar primeiro com o par B através de um par C intermediário, que não está por trás de uma NAT, e com o qual o par B tenha estabelecido uma conexão TCP que está em curso. Então, o par A pode solicitar ao par B, por intermédio do par C, que inicie uma conexão TCP diretamente com ele. Uma vez estabelecida a conexão P2P TCP direta entre os pares A e B, os dois podem trocar mensagens ou arquivos. Essa solução, denominada **reversão de conexão**, na verdade é usada por muitas aplicações P2P para a **travessia de NAT**. Se ambos os pares, A e B, estiverem por trás de suas próprias NATs, a situação é um pouco mais complicada, mas pode ser resolvida utilizando repasses para aplicação, como vimos em repasses do Skype, no Capítulo 2.

UPnP

A travessia da NAT está sendo cada vez mais fornecida pelo *Universal Plug and Play* (UPnP), um protocolo que permite a um hospedeiro descobrir e configurar uma NAT próxima [UPnP Forum, 2012]. O UPnP exige que o hospedeiro e a NAT sejam compatíveis com ele. Com respeito ao UPnP, uma aplicação que roda em um hospedeiro pode solicitar um mapeamento da NAT entre seus (*endereço IP particular, número de porta particular)* e (*endereço IP público, número de porta pública)* para algum número de porta pública requisitado. Se a NAT aceitar a requisição e criar um mapeamento, então nós externos podem iniciar conexões TCP para (*endereço IP público*, *número de porta pública*). Além disso, o UPnP deixa a aplicação saber o valor de (*endereço IP público*, *número de porta pública)*, de modo que ela possa anunciá-los ao mundo exterior.

Como exemplo, suponha que seu hospedeiro, localizado atrás de uma NAT com o UPnP habilitado, possua o endereço IP particular 10.0.0.1 e esteja rodando o BitTorrent na porta 3345. Suponha também que o endereço IP público da NAT seja 138.76.29.7. Sua aplicação BitTorrent decerto quer poder aceitar conexões vindas de outros hospedeiros, para que blocos sejam adquiridos através delas. Para isso, a aplicação BitTorrent, em seu hospedeiro, solicita que a NAT crie uma "abertura" que mapeia (10.0.0.1, 3345) para (138.76.29.7, 5001). (A porta pública número 5001 é escolhida pela aplicação.) A aplicação BitTorrent em seu hospedeiro poderia, também, anunciar a seu rastreador que está disponível em (138.76.29.7, 5001). Dessa maneira, um hospedeiro externo que está rodando o BitTorrent consegue contatar o rastreador e descobrir que sua aplicação BitTorrent está rodando em (138.76.29.7, 5001). O hospedeiro externo pode enviar um pacote SYN TCP para (138.76.29.7, 5001). Quando a NAT receber o pacote SYN, alterará o endereço IP destinatário e o número da porta no pacote para (10.0.0.1, 3345) e encaminhará o pacote pela NAT.

Em resumo, o UPnP permite que hospedeiros externos iniciem sessões de comunicação com hospedeiros que utilizam NAT, usando TCP ou UDP. As NATs foram inimigas das aplicações P2P por um longo tempo; o UPnP, que apresenta uma solução eficaz e potente para a travessia da NAT, pode ser a salvação. Nossa discussão sobre NAT e UPnP foi necessariamente breve. Para obter mais detalhes sobre a NAT, consulte Huston [2004] e Cisco NAT [2012].

4.4.3 Protocolo de Mensagens de Controle da Internet (ICMP)

Lembre-se de que a camada de rede da Internet tem três componentes principais: o protocolo IP, discutido na seção anterior; os protocolos de roteamento da Internet (entre eles RIP, OSPF e BGP), que serão estudados na Seção 4.6; e o ICMP, objeto desta seção.

O ICMP, especificado no [RFC 792], é usado por hospedeiros e roteadores para comunicar informações de camada de rede entre si. A utilização mais comum do ICMP é para comunicação de erros. Por exemplo, ao rodar uma sessão Telnet, FTP ou HTTP, é possível que você já tenha encontrado uma mensagem de erro como "Rede de destino inalcançável". Essa mensagem teve sua origem no ICMP. Em algum ponto, um roteador IP não conseguiu

descobrir um caminho para o hospedeiro especificado em sua aplicação Telnet, FTP ou HTTP. O roteador criou e enviou uma mensagem ICMP do tipo 3 a seu hospedeiro indicando o erro.

O ICMP é com frequência considerado parte do IP, mas, em termos de arquitetura, está logo acima, pois mensagens ICMP são carregadas dentro de datagramas IP. Isto é, mensagens ICMP são carregadas como carga útil IP, exatamente como segmentos TCP ou UDP, que também o são. De maneira semelhante, quando um hospedeiro recebe um datagrama IP com ICMP especificado como protocolo de camada superior, ele demultiplexa o conteúdo do datagrama para ICMP, exatamente como demultiplexaria o conteúdo de um datagrama para TCP ou UDP.

Mensagens ICMP têm um campo de tipo e um campo de código. Além disso, contêm o cabeçalho e os primeiros 8 bytes do datagrama IP que causou a criação da mensagem ICMP em primeiro lugar (de modo que o remetente pode determinar o datagrama que causou o erro). Alguns tipos de mensagens ICMP selecionadas são mostrados na Figura 4.23. Note que mensagens ICMP não são usadas somente para sinalizar condições de erro.

O conhecido programa *ping* envia uma mensagem ICMP do tipo 8 código 0 para o hospedeiro especificado. O hospedeiro de destino, ao ver a solicitação de eco, devolve uma resposta de eco ICMP do tipo 0 código 0. A maior parte das execuções de TCP/IP suporta o servidor ping diretamente no sistema operacional; isto é, o servidor não é um processo. O Capítulo 11 de Stevens [1990] fornece o código-fonte para o programa *ping* cliente. Note que o programa cliente tem de ser capaz de instruir o sistema operacional para que ele gere uma mensagem ICMP do tipo 8 código 0.

Outra mensagem ICMP interessante é a de redução da origem. Essa mensagem é pouco usada na prática. Sua finalidade original era realizar controle de congestionamento — permitir que um roteador congestionado enviasse uma mensagem ICMP de redução da origem a um hospedeiro para obrigá-lo a reduzir sua velocidade de transmissão. Vimos no Capítulo 3 que o TCP tem seu próprio mecanismo de controle de congestionamento, que funciona na camada de transporte sem usar realimentação da camada de rede tal como a mensagem ICMP de redução da origem.

No Capítulo 1, apresentamos o programa Traceroute, que nos permite acompanhar a rota de um hospedeiro a qualquer outro hospedeiro no mundo. O interessante é que o Traceroute é executado com mensagens ICMP. Para determinar os nomes e endereços de roteadores entre a origem e o destino, o Traceroute da origem envia uma série de datagramas comuns ao destino. O primeiro deles tem um TTL de 1, o segundo tem um TTL de 2, o terceiro tem um TTL de 3 e assim por diante. A origem também aciona temporizadores para cada datagrama. Quando o enésimo datagrama chega ao enésimo roteador, este observa que o TTL do datagrama acabou de expirar. Segundo as regras do protocolo IP, o roteador descarta o datagrama

FIGURA 4.23 TIPOS DE MENSAGENS ICMP

Tipo ICMP	Código	Descrição
0	0	resposta de eco (para *ping*)
3	0	rede de destino inalcançável
3	1	hospedeiro de destino inalcançável
3	2	protocolo de destino inalcançável
3	3	porta de destino inalcançável
3	6	rede de destino desconhecida
3	7	hospedeiro de destino desconhecido
4	0	repressão da origem (controle de congestionamento)
8	0	solicitação de eco
9	0	anúncio do roteador
10	0	descoberta do roteador
11	0	TTL expirado
12	0	cabeçalho IP inválido

e envia uma mensagem ICMP de aviso à origem (tipo 11 código 0). Essa mensagem de aviso inclui o nome do roteador e seu endereço IP. Quando chega à origem, a mensagem obtém, do temporizador, o tempo de viagem de ida e volta e, da mensagem ICMP, o nome e o endereço IP do enésimo roteador.

Como uma origem de Traceroute sabe quando parar de enviar segmentos UDP? Lembre-se de que a origem incrementa o campo do TTL para cada datagrama que envia. Assim, um deles conseguirá enfim chegar ao hospedeiro de destino. Como tal datagrama contém um segmento UDP com um número de porta improvável, o hospedeiro de destino devolve à origem uma mensagem ICMP indicando que a porta não pôde ser alcançada (mensagem tipo 3, código 3). Quando recebe essa mensagem ICMP particular, o hospedeiro de origem sabe que não precisa enviar mais pacotes de sondagem. (Na verdade, o programa Traceroute padrão envia conjuntos de três pacotes com o mesmo TTL; assim, a saída de Traceroute oferece três resultados para cada TTL.)

Desse modo, o hospedeiro de origem fica a par do número e das identidades de roteadores que estão entre ele e o hospedeiro de destino e o tempo de viagem de ida e volta entre os dois. Note que o programa cliente Traceroute tem de ser capaz de instruir o sistema operacional para que este gere datagramas UDP com valores específicos de TTL e também tem de poder ser avisado por seu sistema operacional quando chegam mensagens ICMP. Agora que você entende como o Traceroute funciona, é provável que queira voltar e brincar um pouco com ele.

SEGURANÇA EM FOCO

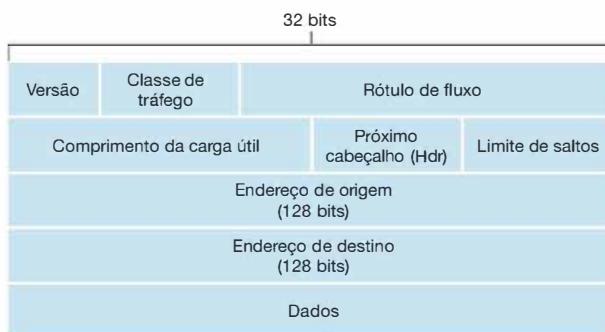
Inspecionando datagramas: *firewalls* e sistemas de detecção de intrusão

Suponha que você deva administrar uma rede doméstica, departamental, acadêmica ou corporativa. Os atacantes, sabendo a faixa de endereço IP da sua rede, podem enviar facilmente datagramas IP para endereços da sua faixa. Tais datagramas são capazes de fazer todos os tipos de coisas desonestas, inclusive mapear sua rede, fazendo seu reconhecimento (*ping sweep*) e varredura de portas, prejudicar hospedeiros vulneráveis com pacotes defeituosos, inundar servidores com milhares de pacotes ICMP e infectar hospedeiros incluindo *malwares* nos pacotes. Como administrador de rede, o que você vai fazer com todos esses vilões capazes de enviar pacotes maliciosos à sua rede? Dois consagrados mecanismos de defesa para esses ataques de pacote são os *firewalls* e os sistemas de detecção de intrusão (IDSs).

Como administrador de rede, você pode, primeiro, tentar instalar um *firewall* entre sua rede e a Internet. (A maioria dos roteadores de acesso, hoje, possui capacidade para *firewall*.) Os *firewalls* inspecionam o datagrama e campos do cabeçalho do segmento, evitando que datagramas suspeitos entrem na rede interna. Um *firewall* pode estar configurado, por exemplo, para bloquear todos os pacotes de requisição de eco ICMP, impedindo assim que um atacante realize um reconhecimento de rede por sua faixa de endereço IP. É capaz também de bloquear pacotes baseando-se nos endereços IP remetente e destinatário e em números de porta. Além disso, os *firewalls* podem ser configurados para rastrear conexões TCP, permitindo a entrada somente de datagramas que pertençam a conexões aprovadas.

Uma proteção adicional pode ser fornecida com um IDS. Um IDS, em geral localizado no limite de rede, realiza "uma inspeção profunda de pacote", examinando não apenas campos de cabeçalho, como também cargas úteis no datagrama (incluindo dados da camada de aplicação). Um IDS possui um banco de dados de assinaturas de pacote à medida que novos ataques são descobertos. Enquanto os pacotes percorrem o IDS, este tenta combinar campos de cabeçalhos e cargas úteis às assinaturas em seu banco de dados de assinaturas. Se tal combinação é encontrada, cria-se um alerta. Os sistemas de prevenção de intrusão (IPS) são semelhantes a um IDS, exceto pelo fato de bloquearem pacotes além de criar alertas. No Capítulo 8, exploraremos mais detalhadamente *firewalls* e IDSs.

Os *firewalls* e os IDSs são capazes de proteger sua rede de todos os ataques? Claro, a resposta é não, visto que os atacantes encontram novas formas de ataques continuamente para os quais as assinaturas ainda não estão disponíveis. Mas os *firewalls* e os IDSs baseados em assinaturas tradicionais são úteis para proteger sua rede de ataques conhecidos.

4.4.4 IPv6

No começo da década de 1990, a IETF (Internet Engineering Task Force) iniciou um esforço para desenvolver o sucessor do protocolo IPv4. Uma motivação importante para isso foi o entendimento de que o espaço de endereços IP de 32 bits estava começando a escassear, com novas sub-redes e nós IP sendo anexados à Internet (e ainda recebendo endereços IP exclusivos) a uma velocidade estonteante. Para atender a essa necessidade de maior espaço para endereços IP, foi desenvolvido um novo protocolo IP, o IPv6. Os projetistas do IPv6 também aproveitaram essa oportunidade para ajustar e ampliar outros aspectos do IPv4, com base na experiência operacional acumulada sobre esse protocolo.

O momento em que todos os endereços IPv4 estariam alocados (e, por conseguinte, mais nenhuma sub-rede poderia ser ligada à Internet) foi objeto de considerável debate. Os dois líderes do grupo de trabalho de Expectativa de Tempo de Vida dos Endereços (Address Lifetime Expectations) da IETF estimaram que os endereços se esgotariam em 2008 e 2018, respectivamente [Solensky, 1996]. Em fevereiro de 2011, a IANA alocou o último conjunto restante de endereços IPv4 a um registrador regional. Embora esses registradores ainda tenham endereços IPv4 disponíveis dentro de seus conjuntos, quando esses endereços se esgotarem, não haverá mais blocos de endereços disponíveis para serem alocados a partir de um conjunto central [Houston, 2011a]. Embora as estimativas de esgotamento de endereço IPv4 de meados de 1990 sugerissem que poderia se passar um longo tempo até que o espaço de endereços do IPv4 fosse esgotado, ficou claro que seria necessário um tempo expressivo para disponibilizar uma nova tecnologia em escala tão gigantesca. Assim, foi dado início ao esforço denominado Próxima Geração do IP (Next Generation IP — IPng) [Bradner, 1996; RFC 1752]. O resultado foi a especificação IP versão 6 (IPv6) [RFC 2460]. (Uma pergunta recorrente é o que aconteceu com o IPv5. Foi proposto de início que o protocolo ST-2 se tornasse o IPv5, porém, mais tarde, esse protocolo foi descartado.) Excelentes fontes de informação sobre o IPv6 podem ser encontradas em Huitema [1998] e IPv6 [2012].

Formato do datagrama IPv6

O formato do datagrama IPv6 é mostrado na Figura 4.24. As mudanças mais importantes introduzidas no IPv6 ficam evidentes no formato do datagrama:

- *Capacidade de endereçamento expandida.* O IPv6 aumenta o tamanho do endereço IP de 32 bits para 128 bits. Isso garante que o mundo não ficará sem endereços IP. Agora, cada grão de areia do planeta pode ter um endereço IP. Além dos endereços para um grupo de individuais, o IPv6 introduziu um novo tipo de endereço, denominado **endereço para qualquer membro do grupo** (*anycast*), que permite que um datagrama seja entregue a qualquer hospedeiro de um grupo. (Essa característica poderia ser usada, por exemplo, para enviar uma mensagem HTTP GET ao site mais próximo de um conjunto de sites espelhados que contenham um dado documento.)
- *Cabeçalho aprimorado de 40 bytes.* Como discutiremos adiante, vários campos IPv4 foram descartados ou tornaram-se opcionais. O cabeçalho de comprimento fixo de 40 bytes resultante permite processamento mais veloz do datagrama IP. Uma nova codificação de opções permite um processamento de opções mais flexível.
- *Rotulação de fluxo e prioridade.* O IPv6 tem uma definição dúbia de **fluxo**. O RFC 1752 e o RFC 2460 declaram que isso permite "rotular pacotes que pertencem a fluxos particulares para os quais o remetente requisita tratamento especial, tal como um serviço de qualidade não padrão ou um serviço de tempo real". Por exemplo, a transmissão de áudio e vídeo seria tratada como um fluxo. Por outro lado, aplicações mais tradicionais, como transferência de arquivos e e-mail, poderiam não ser tratadas assim. É possível que o tráfego carregado por um usuário de alta prioridade (digamos, alguém que paga por um serviço melhor de tráfego) seja também tratado como um fluxo. O que fica claro, contudo, é que os projetistas do IPv6 preveem a possível necessidade de conseguir diferenciá-los, mesmo que o exato significado de fluxo ainda não tenha sido determinado. O cabeçalho IPv6 também tem um

FIGURA 4.24 FORMATO DO DATAGRAMA IPv6

campo de 8 bits para classe de tráfego. Assim, como o campo TOS do IPv4, ele pode ser usado para dar prioridade a certos datagramas em um fluxo ou a datagramas de certas aplicações (por exemplo, pacotes ICMP) em relação aos de outras (por exemplo, notícias pela rede).

Como foi observado anteriormente, uma comparação entre as figuras 4.24 e 4.13 revela uma estrutura mais simples e mais aprimorada para o datagrama IPv6. Os seguintes campos são definidos no IPv6:

- *Versão*. Esse campo de 4 bits identifica o número da versão do IP. Não é surpresa que o IPv6 tenha o valor 6. Note que colocar 4 nesse campo não cria um datagrama IPv4 válido. (Se criasse, a vida seria bem mais simples — veja a discussão mais adiante, referente à transição do IPv4 para o IPv6.)
- *Classe de tráfego*. Esse campo de 8 bits tem função semelhante à do campo TOS que vimos no IPv4.
- *Rótulo de fluxo*. Como já discutimos, esse campo de 20 bits é usado para identificar um fluxo de datagramas.
- *Comprimento da carga útil*. Esse valor de 16 bits é tratado como um número inteiro sem sinal que dá o número de bytes no datagrama IPv6 que se segue ao pacote do cabeçalho, que tem tamanho fixo de 40 bytes.
- *Próximo cabeçalho*. Esse campo identifica o protocolo ao qual o conteúdo (campo de dados) desse datagrama será entregue (por exemplo, TCP ou UDP). Usa os mesmos valores do campo de protocolo no cabeçalho IPv4.
- *Limite de saltos*. O conteúdo desse campo é decrementado em um para cada roteador que repassa o datagrama. Se a contagem do limite de saltos chegar a zero, o datagrama será descartado.
- *Endereços de origem e de destino*. Os vários formatos do endereço de 128 bits do IPv6 são descritos no RFC 4291.
- *Dados*. Esta é a parte da carga útil do datagrama IPv6. Quando este alcança seu destino, a carga útil pode ser extraída do datagrama IP e passada adiante para o protocolo especificado no campo de próximo cabeçalho.

Nessa discussão, apresentamos a finalidade dos campos que estão incluídos no datagrama IPv6. Quando comparamos o formato do datagrama IPv6 da Figura 4.24 com o formato do datagrama IPv4 que vimos na Figura 4.13, notamos que diversos campos que aparecem no datagrama IPv4 não estão presentes no datagrama IPv6:

- *Fragmentação/remontagem*. O IPv6 não permite fragmentação e remontagem em roteadores intermediários; essas operações podem ser realizadas apenas pela origem e pelo destino. Se um datagrama IPv6 recebido por um roteador for muito grande para ser repassado pelo enlace de saída, o roteador apenas descartará o datagrama e devolveráao remetente uma mensagem de erro ICMP "Pacote muito grande" (veja a seguir). O remetente pode então reenviar os dados usando um datagrama IP de tamanho menor.

Fragmentação e remontagem são operações que tomam muito tempo; retirar essas funcionalidades dos roteadores e colocá-las nos sistemas finais acelera consideravelmente o repasse IP para dentro da rede.

- *Soma de verificação do cabeçalho*. Como os protocolos de camada de transporte (por exemplo, TCP e UDP) e de enlace de dados (por exemplo, Ethernet) nas camadas da Internet realizam soma de verificação, os projetistas do IP provavelmente acharam que essa funcionalidade era tão redundante na camada de rede que podia ser retirada. Mais uma vez, o processamento rápido de pacotes IP era uma preocupação principal. Lembre-se de que em nossa discussão sobre o IPv4 na Seção 4.4.1 vimos que, como o cabeçalho IPv4 contém um campo TTL (semelhante ao campo de limite de saltos no IPv6), a soma de verificação do cabeçalho IPv4 precisava ser recalculada em cada roteador. Como acontece com a fragmentação e a remontagem, essa também era uma operação de alto custo no IPv4.
- *Opções*. O campo de opções não faz mais parte do cabeçalho-padrão do IP. Contudo, ele ainda não saiu de cena. Em vez disso, passou a ser um dos possíveis próximos cabeçalhos a ser apontados pelo cabeçalho do IPv6. Ou seja, assim como os cabeçalhos dos protocolos TCP e UPD podem ser o próximo dentro de um pacote IP, o campo de opções também poderá ser. A remoção do campo de opções resulta em um cabeçalho IP de tamanho fixo de 40 bytes.

Lembre-se de que dissemos na Seção 4.4.3 que o protocolo ICMP é usado pelos nós IP para informar condições de erro e fornecer informações limitadas (por exemplo, a resposta de eco para uma mensagem ping) a um sistema final. Uma nova versão do ICMP foi definida para o IPv6 no RFC 4443. Além da reorganização das definições existentes de tipos e códigos ICMP, o ICMPv6 adicionou novos tipos e códigos exigidos pela nova funcionalidade do IPv6. Entre eles estão incluídos o tipo "Pacote muito grande" e um código de erro "opções IPv6 não reconhecidas". Além disso, o ICMPv6 incorpora a funcionalidade do IGMP (Internet Group Management Protocol — protocolo de gerenciamento de grupos da Internet), que estudaremos na Seção 4.7. O IGMP, que é usado para gerenciar a adesão ou a saída de um hospedeiro de grupos *multicast*, era anteriormente um protocolo separado do ICMP no IPv4.

Transição do IPv4 para o IPv6

Agora que vimos os detalhes técnicos do IPv6, vamos tratar de um assunto muito prático: como a Internet pública, que é baseada no IPv4, fará a transição para o IPv6? O problema é que, enquanto os novos sistemas habilitados para IPv6 podem ser compatíveis, isto é, podem enviar, rotear e receber datagramas IPv4, os sistemas habilitados para IPv4 não podem manusear datagramas IPv6. Há várias opções possíveis [Huston 2011b].

Uma opção seria determinar um "dia da conversão" — uma data e um horário definidos em que todas as máquinas da Internet seriam desligadas e atualizadas, passando do IPv4 para o IPv6. A última transição importante de tecnologia (do uso do NCP para o uso do TCP para serviço confiável de transporte) ocorreu há quase 25 anos. E, mesmo naquela época [RFC 801], quando a Internet era pequenina e ainda gerenciada por um número reduzido de "sabichões", ficou claro que esse "dia da conversão" não era possível. Um dia assim, envolvendo centenas de milhões de máquinas e milhões de administradores e usuários de rede, é ainda mais impensável hoje. O RFC 4213 descreve duas abordagens (que podem ser usadas independentemente ou em conjunto) para a integração gradual dos hospedeiros e roteadores IPv4 ao mundo IPv6 (com a meta de longo prazo de fazer a transição de todos os nós IPv4 para IPv6).

Provavelmente, a maneira mais direta de introduzir nós habilitados ao IPv6 seja uma abordagem de **pilha dupla**, em que nós IPv6 também tenham uma implementação IPv4 completa. Esse nó, denominado nó IPv6/IPv4 no RFC 4213, estaria habilitado a enviar e receber datagramas tanto IPv4 quanto IPv6. Ao interagir com um nó IPv4, um nó IPv6/IPv4 poderá usar datagramas IPv4; ao interagir com um nó IPv6, poderá utilizar IPv6. Nós IPv6/IPv4 devem ter endereços IPv6 e IPv4. Além disso, devem ser capazes de determinar se outro nó é habilitado para IPv6 ou apenas para IPv4. Esse problema pode ser resolvido usando o DNS (veja o Capítulo 2), que poderá retornar um endereço IPv6 se o nome do nó a ser resolvido for capacitado para IPv6. Caso contrário,

ele retornará um endereço IPv4. É claro que, se o nó que estiver emitindo a requisição DNS for habilitado apenas para IPv4, o DNS retornará apenas um endereço IPv4.

Na abordagem de pilha dupla, se o remetente ou o destinatário forem habilitados apenas para IPv4, um datagrama IPv4 deverá ser usado. Como resultado, é possível que dois nós habilitados para IPv6 acabem, basicamente, enviando datagramas IPv4 um para o outro. Isso é ilustrado na Figura 4.25. Suponha que o nó A utiliza IPv6 e queira enviar um datagrama IP ao nó *F*, que também utiliza IPv6. Os nós *A* e *B* podem trocar um datagrama IPv6. Contudo, o nó *B* deve criar um datagrama IPv4 para enviar a *C*. É claro que o campo de dados do datagrama IPv6 pode ser copiado para o do datagrama IPv4 e que o mapeamento adequado de endereço também pode ser feito. No entanto, ao realizar a conversão de IPv6 para IPv4, haverá campos IPv6 específicos no datagrama IPv6 (por exemplo, o campo do identificador de fluxo) que não terão contrapartes em IPv4. As informações contidas nesses campos serão perdidas. Assim, mesmo que *E* e *F* possam trocar datagramas IPv6, os datagramas IPv4 que chegarem a *E* e *D* não conterão todos os campos que estavam no datagrama IPv6 original enviado de *A*.

Uma alternativa para a abordagem de pilha dupla, também discutida no RFC 4213, é conhecida como **implantação de túnel**. O túnel pode resolver o problema observado antes, permitindo, por exemplo, que *E* receba o datagrama IPv6 originado por *A*. A ideia básica por trás da implementação do túnel é a seguinte. Suponha que dois nós IPv6 (por exemplo, *B* e *E* na Figura 4.25) queiram interagir usando datagramas IPv6, mas estão conectados um ao outro por roteadores intervenientes IPv4. Referimo-nos ao conjunto de roteadores intervenientes IPv4 entre dois roteadores IPv6 como um **túnel**, como ilustrado na Figura 4.26. Com a implementação do túnel, o nó IPv6 no lado remetente do túnel (por exemplo, *B*) pega o datagrama IPv6 *inteiro* e o coloca no campo de dados (carga útil) de um datagrama IPv4. Esse datagrama IPv4 é então endereçado ao nó IPv6 no lado receptor (por exemplo, *E*) e enviado ao primeiro nó do túnel (por exemplo, *C*). Os roteadores IPv4 intermediários o direcionam entre eles, exatamente como fariam com qualquer outro, alheios ao fato de que o datagrama IPv4 contém um datagrama IPv6 completo. O nó IPv6 do lado receptor do túnel por fim recebe o datagrama IPv4 (pois ele é o destino do datagrama IPv4!), determina que ele contém um datagrama IPv6, extrai este último e, então, o roteia exatamente como faria se tivesse recebido o datagrama IPv6 de um vizinho IPv6 diretamente ligado a ele.

Encerramos esta seção mencionando que enquanto a adoção do IPv6 estava demorando para decolar [Lawton, 2001], a iniciativa foi tomada recentemente. Consulte Huston [2008b] para o emprego do IPv6 a partir de 2008; veja em NIST IPv6 [2012] um instantâneo da implantação do IPv6 nos Estados Unidos. A proliferação de dispositivos como telefones e outros equipamentos portáteis preparados para IP dá um empurrão extra para a implementação geral do IPv6. O Third Generation Partnership Program [3GPP 2012] na Europa especificou o IPv6 como o esquema de endereçamento padrão para multimídia em dispositivos móveis.

Uma lição importante que podemos aprender com a experiência do IPv6 é que há enorme dificuldade para mudar protocolos de camada de rede. Desde o início da década de 1990, numerosos novos protocolos foram anunciados como a próxima maior revolução da Internet, mas a maioria deles teve pouca aceitação até agora. Dentre esses, estão o IPv6, os protocolos de grupo (ou *multicast*, veja a Seção 4.7) e os protocolos de reserva de

FIGURA 4.25 ABORDAGEM DE PILHA DUPLA

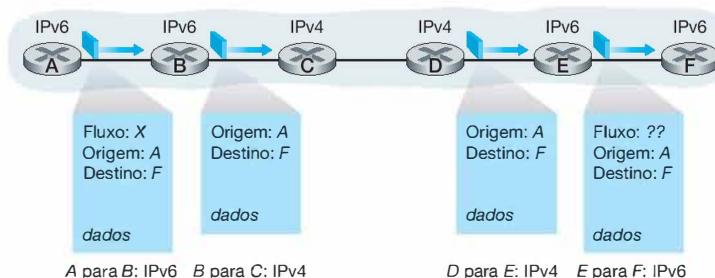
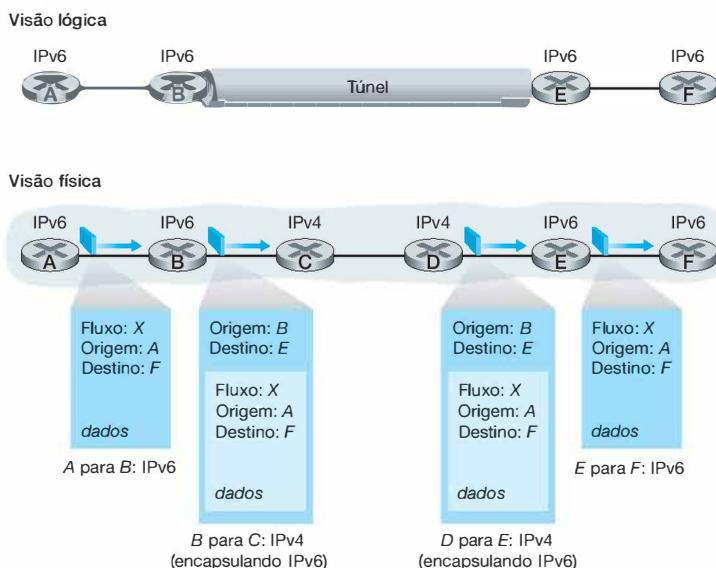



FIGURA 4.26 IMPLEMENTAÇÃO DE TÚNEL

recursos (Capítulo 7). Na verdade, introduzir novos protocolos na camada de rede é como substituir os alicerces de uma casa — é difícil de fazer sem demolir a casa inteira ou, no mínimo, retirar os moradores temporariamente da residência. Por outro lado, a Internet vem testemunhando a rápida disponibilização de novos protocolos na camada de aplicação. Os exemplos clássicos são, é claro, a Web, mensagens instantâneas e o compartilhamento de arquivos P2P. Outros exemplos são a recepção de áudio e vídeo em tempo real e os jogos distribuídos. Introduzir novos protocolos de camada de aplicação é como acrescentar uma nova camada de tinta em uma casa — é relativamente fácil de fazer e, se você escolher uma cor atraente, outras casas da vizinhança vão imitá-lo. Em resumo, no futuro podemos esperar mudanças na camada de rede da Internet, mas elas provavelmente ocorrerão dentro de uma escala de tempo bem mais lenta do que as que acontecerão na camada de aplicação.

4.4.5 Uma breve investida em segurança IP

Na Seção 4.4.3 discutimos o IPv4 com alguns detalhes, incluindo os serviços que oferece e como são implementados. Ao ler esta seção, você pode ter percebido que nenhum serviço de segurança foi mencionado. De fato, o IPv4 foi projetado em uma era (anos 1970) em que a Internet era utilizada, principalmente, entre pesquisadores de rede mutuamente confiáveis. Criar uma rede de computadores que integrava uma grande quantidade de tecnologias da camada de enlace já era desafiador, sem ter de se preocupar com a segurança.

Contudo, com a segurança sendo a principal preocupação hoje, pesquisadores da Internet começaram a criar novos protocolos da camada de rede que oferecem variados serviços de segurança. Um desses protocolos é o IPsec, um dos protocolos da camada de rede mais conhecidos e empregados em Redes Virtuais Privadas (VPNs). Embora o IPsec e seus suportes sejam abordados no Capítulo 8, apresentamos uma breve introdução sobre seus serviços nesta seção.

O IPsec foi desenvolvido para ser compatível com o IPv4 e o IPv6. Em particular, para obter os benefícios do IPv6, não precisamos substituir as pilhas dos protocolos em *todos* os roteadores e hospedeiros na Internet. Por exemplo, usando o modo transporte (um dos "modos" do IPsec), se dois hospedeiros querem se comunicar em segurança, o IPsec precisa estar disponível apenas nesses dois. Todos os outros roteadores e hospedeiros podem continuar a rodar o IPv4 simples.

Para uma abordagem concreta, a partir daqui focaremos no modo de transporte do IPsec. Nesse modo, dois hospedeiros estabelecem, primeiro, uma sessão IPsec entre si mesmos. (Assim, o IPsec é orientado a conexão!) Com a sessão pronta, todos os segmentos TCP e UDP enviados entre os dois hospedeiros aproveitam os serviços de segurança fornecidos pelo IPsec. No lado remetente, a camada de transporte passa um segmento para o IPsec. Este, então, codifica o segmento, acrescenta campos de segurança adicionais a ele e envolve a carga útil resultante em um datagrama IP comum. (Na verdade, isso é um pouco mais complicado, como veremos no Capítulo 8.) Depois, o hospedeiro remetente envia o datagrama para a Internet, a qual o transporta ao destinatário. Em seguida, o IPsec decodifica o segmento e o encaminha à camada de transporte.

Os serviços oferecidos por uma sessão IPsec incluem:

- *Acordo criptográfico.* Mecanismos que permitem que dois hospedeiros de comunicação concordem nos algoritmos criptográficos e chaves.
- *Codificação das cargas úteis do datagrama IP.* Quando o hospedeiro destinatário recebe um segmento da camada de transporte, o IPsec codifica a carga útil, que pode ser somente decodificada pelo IPsec no hospedeiro destinatário.
- *Integridade dos dados.* O IPsec permite que o hospedeiro destinatário verifique se os campos do cabeçalho do datagrama e a carga útil codificada não foram modificados enquanto o datagrama estava no caminho da origem ao destino.
- *Autenticação de origem.* Quando um hospedeiro recebe um datagrama IPsec de uma origem confiável (com uma chave confiável — veja no Capítulo 8), o hospedeiro está certo de que o endereço IP remetente no datagrama é a verdadeira origem do datagrama.

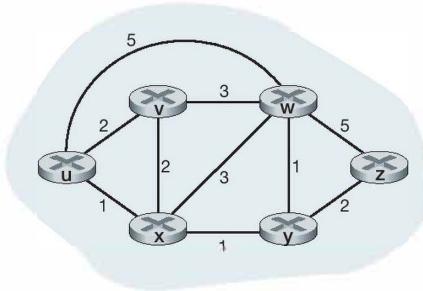
Quando dois hospedeiros estabelecem uma sessão IPsec, todos os segmentos TCP e UDP enviados entre eles serão codificados e autenticados. O IPsec, portanto, oferece uma cobertura geral, protegendo toda a comunicação entre os dois hospedeiros para todas as aplicações de rede.

Uma empresa pode utilizar o IPsec para se comunicar de forma segura na Internet pública não segura. Para fins de ilustração, verificaremos um exemplo simples. Considere uma empresa que possua um grande número de vendedores que viajam, cada um levando um notebook da companhia. Suponha que os vendedores precisem consultar, com frequência, informações confidenciais sobre a empresa (por exemplo, sobre preços e produtos) armazenadas em um servidor na matriz. Imagine, ainda, que os vendedores também precisem enviar documentos confidenciais um para o outro. Como isso pode ser feito com o IPsec? Como você pode imaginar, instalamos o IPsec no servidor e em todos os notebooks dos vendedores. Com ele instalado nesses hospedeiros, quando um vendedor precisar se comunicar com o servidor ou com outro vendedor, a sessão de comunicação estará protegida.

4.5 ALGORITMOS DE ROTEAMENTO

Exploramos, neste capítulo, a função de repasse da camada de rede mais do que qualquer outra. Aprendemos que, quando um pacote chega a um roteador, este indexa uma tabela de repasse e determina a interface de enlace para a qual o pacote deve ser dirigido. Aprendemos também que algoritmos de roteamento que rodam em roteadores de rede trocam e calculam as informações que são utilizadas para configurar essas tabelas de repasse. A interação entre algoritmos de roteamento e tabelas de repasse foi ilustrada na Figura 4.2. Agora que já nos aprofundamos um pouco na questão do repasse, voltaremos a atenção para o outro tópico importante desse capítulo, a saber, a função crítica da camada de rede, o roteamento. Quer ofereça um serviço de datagramas (quando pacotes diferentes entre um determinado par origem-destino podem seguir rotas diferentes) ou um serviço de VCs (quando todos os pacotes entre uma origem e um destino determinados pegarão o mesmo caminho), a camada de rede deve, mesmo assim, determinar o caminho que os pacotes percorrem entre remetentes e destinatários. Veremos que a tarefa do roteamento é determinar bons caminhos (ou rotas) entre remetentes e destinatários através da rede de roteadores.

Em geral um hospedeiro está ligado diretamente a um roteador, o **roteador *default*** para esse hospedeiro (também denominado **roteador do primeiro salto**). Sempre que um hospedeiro emitir um pacote, o pacote será transferido para seu roteador *default*. Denominamos **roteador de origem** o roteador *default* do hospedeiro de origem e **roteador de destino** o roteador *default* do hospedeiro de destino. O problema de rotear um pacote do hospedeiro de origem até o hospedeiro de destino se reduz, claramente, ao problema de direcionar o pacote do roteador de origem ao roteador de destino, que é o foco desta seção.

Portanto, a finalidade de um algoritmo de roteamento é simples: dado um conjunto de roteadores conectados por enlaces, um algoritmo de roteamento descobre um "bom" caminho entre o roteador de origem e o de destino. Em geral, um "bom" caminho é aquele que tem o "menor custo". No entanto, veremos que, na prática, preocupações do mundo real, como questões de política (por exemplo, uma regra que determina que "o roteador *x*, de propriedade da organização *Y*, não deverá repassar nenhum pacote originário da rede de propriedade da organização *Z*"), também entram em jogo para complicar algoritmos conceitualmente simples e elegantes, cuja teoria fundamenta a prática de roteamento nas redes de hoje.

Um grafo é usado para formular problemas de roteamento. Lembre-se de que um **grafo** $G = (N,E)$ é um conjunto N de nós e uma coleção E de arestas, no qual cada aresta é um par de nós do conjunto N. No contexto do roteamento da camada de rede, os nós do grafo representam roteadores — os pontos nos quais são tomadas decisões de repasse de pacotes — e as arestas que conectam os nós representam os enlaces físicos entre esses roteadores. Uma abstração gráfica de uma rede de computadores está exibida na Figura 4.27. Para ver alguns grafos representando mapas de rede reais, consulte Dodge [2012]; Cheswick [2000]; para uma discussão de como os diferentes modelos baseados em grafo modelam a Internet, consulte Zegura [1997]; Faloutsos [1999]; Li [2004].

Como ilustrado na Figura 4.27, uma aresta também tem um valor que representa seu custo. Em geral, o custo de uma aresta pode refletir o tamanho físico do enlace correspondente (por exemplo, um enlace transoceânico poderia ter um custo mais alto do que um enlace terrestre de curta distância), a velocidade do enlace ou o custo monetário a ele associado. Para nossos objetivos, consideraremos os custos da aresta apenas como um dado e não nos preocuparemos com o modo como eles são determinados. Para qualquer aresta (x, y) em E, denominamos $c(x, y)$ o custo da aresta entre os nós x e y. Se o par (x, y) não pertencer a E, estabelecemos $c(x, y) = \infty$. Além disso, sempre consideraremos somente grafos não direcionados (isto é, grafos cujas arestas não têm uma direção), de modo que a aresta (x,y) é a mesma que a aresta (y,x) e $c(x,y) = c(y,x)$. Dizemos também que y é um **vizinho** do nó x se (x,y) pertencer a E.

Dado que são atribuídos custos às várias arestas na abstração do grafo, uma meta natural de um algoritmo de roteamento é identificar o caminho de menor custo entre origens e destinos. Para tornar esse problema mais preciso, lembremos que um **caminho** em um grafo $G = (N,E)$ é uma sequência de nós $(x_1, x_2,..., x_p)$ tal que cada um dos pares $(x_1, x_2), (x_2, x_3),...,(x_{p-1}, x_p)$ são arestas em E. O custo de um caminho $(x_1, x_2,..., x_p)$ é apenas a soma de todos os custos das arestas ao longo do caminho, ou seja, $c(x_1, x_2) + c(x_2, x_3) + ... + c(x_{p-1},x_p)$. Dados quaisquer dois nós x e y, em geral há muitos caminhos entre os dois, e cada caminho tem um custo. Um ou mais desses caminhos é um **caminho de menor custo**. Por conseguinte, o problema do menor custo é claro: descobrir um caminho entre a origem e o destino que tenha o menor custo. Na Figura 4.27, por exemplo, o caminho de menor custo entre o nó da origem u e o de destino w é (u, x, y, w), cujo custo de caminho é 3. Note que, se todas as arestas do grafo tiverem o mesmo custo, o caminho de menor custo também é o **caminho mais curto** (isto é, o que tem o menor número de enlaces entre a origem e o destino).

Apenas como simples exercício, tente descobrir o caminho de menor custo entre os nós u e z na Figura 4.27 e reflita um pouco sobre como você o calculou. Se você for como a maioria das pessoas, descobriu o caminho de u a z examinando a figura, traçando algumas rotas de u a z, e se convencendo, de algum modo, que o caminho escolhido tinha o menor custo entre todos os possíveis. (Você verificou todos os 17 possíveis caminhos entre u e z? Provavelmente, não!) Esse cálculo é um exemplo de um algoritmo de roteamento centralizado — o algoritmo é rodado em um local, o seu cérebro, com informações completas sobre a rede. De modo geral, uma maneira possível de classificar algoritmos de roteamento é como globais ou centralizados.

FIGURA 4.27 MODELO ABSTRATO DE GRAFO DE UMA REDE DE COMPUTADORES

- Um **algoritmo de roteamento global** calcula o caminho de menor custo entre uma origem e um destino usando conhecimento completo e global sobre a rede. Em outras palavras, o algoritmo considera como entradas a conectividade entre todos os nós e todos os custos dos enlaces. E isso exige que o algoritmo obtenha essas informações, de algum modo, antes de realizar de fato o cálculo. Este pode ser rodado em um local (um algoritmo de roteamento global centralizado) ou replicado em vários locais. Contudo, a principal característica distintiva, nesse caso, é que um algoritmo global tem informação completa sobre conectividade e custo de enlaces. Na prática, algoritmos com informação global de estado são com frequência denominados **algoritmos de estado de enlace** (*link-state* — **LS**), já que devem estar a par dos custos de cada enlace na rede. Estudaremos algoritmos de estado de enlace na Seção 4.5.1.
- Em um **algoritmo de roteamento descentralizado**, o cálculo do caminho de menor custo é realizado de modo iterativo e distribuído. Nenhum nó tem informação completa sobre os custos de todos os enlaces da rede. Em vez disso, cada nó começa sabendo apenas os custos dos enlaces diretamente ligados a ele. Então, por meio de um processo iterativo de cálculo e de troca de informações com seus nós vizinhos (isto é, que estão na outra extremidade dos enlaces aos quais ele próprio está ligado), um nó gradualmente calcula o caminho de menor custo até um destino ou um conjunto de destinos. O algoritmo de roteamento descentralizado que estudaremos logo adiante na Seção 4.5.2 é denominado algoritmo de vetor de distâncias (*distance-vector algorithm* — DV), porque cada nó mantém um vetor de estimativas de custos (distâncias) de um nó até todos os outros nós da rede.

Uma segunda maneira geral de classificar algoritmos de roteamento é como estáticos ou dinâmicos. Em **algoritmos de roteamento estáticos**, as rotas mudam muito devagar ao longo do tempo, muitas vezes como resultado de intervenção humana (por exemplo, uma pessoa editando manualmente a tabela de repasse do roteador). **Algoritmos de roteamento dinâmicos** mudam os caminhos de roteamento à medida que mudam as cargas de tráfego ou a topologia da rede. Um algoritmo dinâmico pode ser rodado periodicamente ou como reação direta a mudanças de topologia ou de custo dos enlaces. Ao mesmo tempo em que são mais sensíveis a mudanças na rede, algoritmos dinâmicos também são mais suscetíveis a problemas como *loops* de roteamento e oscilação em rotas.

Uma terceira maneira de classificar algoritmos de roteamento é como sensíveis à carga ou insensíveis à carga. Em um **algoritmo sensível à carga**, custos de enlace variam dinamicamente para refletir o nível corrente de congestionamento no enlace subjacente. Se houver um alto custo associado com um enlace que está congestionado, um algoritmo de roteamento tenderá a escolher rotas que evitem esse enlace. Embora antigos algoritmos de roteamento da ARPAnet fossem sensíveis à carga [McQuillan, 1980], foram encontradas várias dificuldades [Huitema, 1998]. Os algoritmos de roteamento utilizados na Internet hoje (como RIP, OSPF e BGP) são **insensíveis à carga**, pois o custo de um enlace não reflete explicitamente seu nível de congestionamento atual (nem o mais recente).

4.5.1 O algoritmo de roteamento de estado de enlace (LS)

Lembre-se de que, em um algoritmo de estado de enlace, a topologia da rede e todos os custos de enlace são conhecidos, isto é, estão disponíveis como dados para o algoritmo de estado de enlace. Na prática, isso se consegue fazendo cada nó transmitir pacotes de estado de enlace a *todos* os outros nós da rede, uma vez que cada um desses pacotes contém as identidades e os custos dos enlaces ligados a ele. Na prática (por exemplo, com o protocolo de roteamento OSPF da Internet, discutido na Seção 4.6.1) isso frequentemente é conseguido com um algoritmo de **transmissão por difusão de estado de enlace** [Perlman, 1999]. Algoritmos de transmissão por difusão serão estudados na Seção 4.7. O resultado da transmissão por difusão dos nós é que todos os nós têm uma visão idêntica e completa da rede. Cada um pode, então, rodar o algoritmo de estado de enlace e calcular o mesmo conjunto de caminhos de menor custo como todos os outros nós.

O algoritmo de roteamento de estado de enlace que apresentamos adiante é conhecido como *algoritmo de Dijkstra*, o nome de seu inventor. Um algoritmo que guarda relações muito próximas com ele é o algoritmo de Prim; consulte Cormen [2001] para ver uma discussão geral sobre algoritmos de grafo. O algoritmo de Dijkstra calcula o caminho de menor custo entre um nó (a origem, que chamaremos de *u*) e todos os outros nós da rede. É um algoritmo iterativo e tem a propriedade de, após a k-ésima iteração, conhecer os caminhos de menor custo para *k* nós de destino e, dentre os caminhos de menor custo até todos os nós de destino, esses *k* caminhos terão os *k* menores custos. Vamos definir a seguinte notação:

- *D(v)*: custo do caminho de menor custo entre o nó de origem e o destino *v* até essa iteração do algoritmo.
- *p(v)*: nó anterior (vizinho de *v*) ao longo do caminho de menor custo corrente desde a origem até *v*.
- *N'*: subconjunto de nós; *v* pertence a *N'* se o caminho de menor custo entre a origem e *v* for inequivocamente conhecido.

O algoritmo de roteamento global consiste em uma etapa de inicialização seguida de um *loop*. O número de vezes que o *loop* é rodado é igual ao número de nós na rede. Ao terminar, o algoritmo terá calculado os caminhos mais curtos desde o nó de origem *u* até cada um dos outros nós da rede.

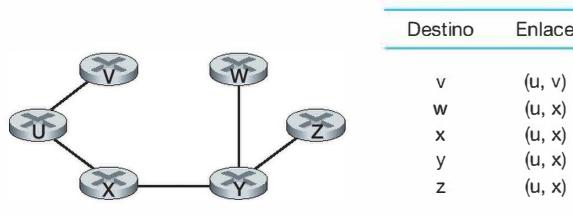
Algoritmo de estado de enlace para o nó de origem *u*

```
1  Inicialização
2      N' = {u}
3      para todos os nós v
4          se v for um vizinho de u
5              então D(v) = c(u,v)
6          senão D(v) = ∞
7
8  Loop
9      encontre w não em N' tal que D(w) é um mínimo
10     adicione w a N'
11     atualize D(v) para cada vizinho v de w e não em N':
12             D(v) = min( D(v), D(w) + c(w,v) )
13     /* o novo custo para v é o velho custo para v ou
14         o custo do menor caminho conhecido para w mais o custo de w para v */
15 até N'= N
```

Como exemplo, vamos considerar a rede da Figura 4.27 e calcular os caminhos de menor custo de *u* até todos os destinos possíveis. Os cálculos do algoritmo estão resumidos na Tabela 4.3, na qual cada linha fornece os valores das variáveis do algoritmo ao final da iteração. Vamos examinar detalhadamente alguns dos primeiros estágios:

- No estágio de inicialização, os caminhos de menor custo mais conhecidos de *u* até os vizinhos diretamente ligados a ele (*v*, *w* e *x*) são inicializados em 2, 1 e 5, respectivamente. Note, em particular, que o custo até *w* é estabelecido em 5 (embora logo veremos que, na realidade, existe um trajeto cujo custo é ainda

menor), já que este é o custo do enlace (um salto) direto *u* a *w*. Os custos até *y* e *z* são estabelecidos como infinito, porque eles não estão diretamente conectados a *u*.

- Na primeira iteração, examinamos os nós que ainda não foram adicionados ao conjunto *N'* e descobrimos o nó de menor custo ao final da iteração anterior. Este é o nó *x*, com um custo de 1, e, assim, *x* é adicionado ao conjunto *N'*. A linha 12 do algoritmo de vetor de distâncias (LS) é então rodada para atualizar *D*(*v*) para todos os nós *v*, produzindo os resultados mostrados na segunda linha (Etapa 1) da Tabela 4.3. O custo do caminho até *v* não muda. Descobriremos que o custo do caminho até *w* pelo nó *x* (que era 5 ao final da inicialização) é 4. Por conseguinte, esse caminho de custo mais baixo é selecionado e o predecessor de *w* ao longo do caminho mais curto a partir de *u* é definido como *x*. De maneira semelhante, o custo até *y* (através de *x*) é calculado como 2 e a tabela é atualizada de acordo com isso.
- Na segunda iteração, verificamos que os nós *v* e *y* são os que têm os caminhos de menor custo (2); decidimos o empate arbitrariamente e adicionamos *y* ao conjunto *N'* de modo que *N'* agora contém *u*, *x* e *y*. O custo dos nós remanescentes que ainda não estão em *N'* (isto é, nós *v*, *w* e *z*) são atualizados pela linha 12 do algoritmo LS, produzindo os resultados mostrados na terceira linha da Tabela 4.3.
- E assim por diante…

Quando o algoritmo LS termina, temos, para cada nó, seu predecessor ao longo do caminho de menor custo a partir do nó de origem. Temos também *o predecessor* para cada um deles; assim, podemos construir o caminho inteiro da origem até todos os destinos. Então, a tabela de repasse em um nó, por exemplo, *u*, pode ser construída a partir dessas informações, armazenando, para cada destino, o nó do salto seguinte no caminho de menor custo de *u* até o destino. A Figura 4.28 mostra os caminhos de menor custo resultantes e a tabela de repasse em *u* para a rede na Figura 4.27.

Qual é a complexidade do cálculo desse algoritmo? Isto é, dados *n* nós (sem contar a origem), quanto cálculo é preciso efetuar no pior caso para descobrir os caminhos de menor custo entre a origem e todos os destinos? Na primeira iteração, precisamos pesquisar todos os *n* nós para determinar o nó *w*, que não está em *N'*, e que tem o custo mínimo. Na segunda, temos de verificar $n - 1$ nós para determinar o custo mínimo. Na terceira, $n - 2$ nós. E assim por diante. Em termos gerais, o número total de nós que precisamos pesquisar em todas as iterações é $n(n + 1)/2$, e, assim, dizemos que a complexidade da implementação do algoritmo de estado de enlace para o pior caso é de ordem *n* ao quadrado: $O(n^2)$. (Uma execução mais sofisticada, que utiliza uma estrutura de dados conhecida como pilha, pode descobrir o mínimo na linha 9 em tempo logarítmico e não linear, reduzindo assim a complexidade.)

Antes de concluirmos nossa discussão sobre o algoritmo LS, vamos considerar uma patologia que pode surgir. A Figura 4.29 mostra uma topologia de rede simples em que os custos dos enlaces são iguais à carga transportada pelo enlace, refletindo, por exemplo, o atraso que seria experimentado. Nesse exemplo, os custos dos enlaces não são simétricos, isto é, $c(u,v)$ é igual a $c(v,u)$ apenas se a carga transportada em ambas as direções do enlace (u,v) for a mesma. Nesse exemplo, o nó *z* origina uma unidade de tráfego destinada a *w*, o nó *x* também origina uma unidade de tráfego destinada a *w* e o nó *y* injeta uma quantidade de tráfego igual a *e*, também destinada a *w*.

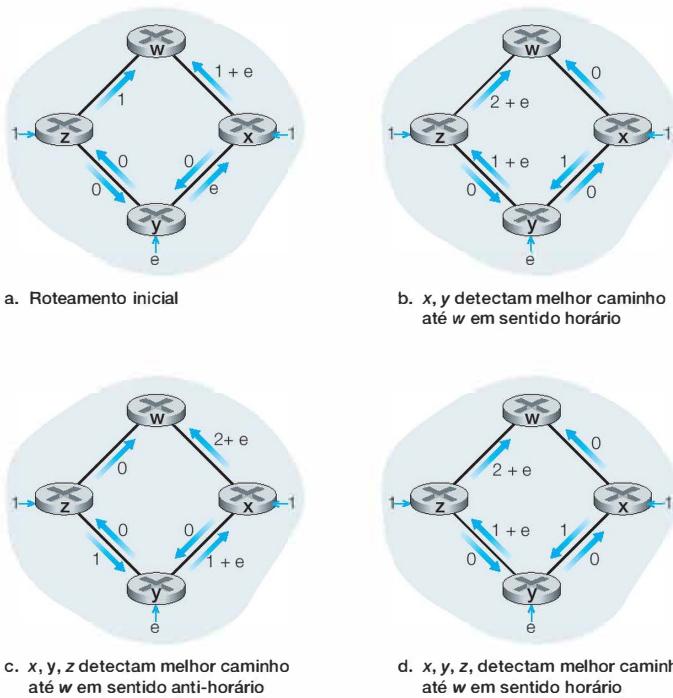
TABELA 4.3 EXECUÇÃO DO ALGORITMO DE ESTADO DE ENLACE NA REDE DA FIGURA 4.27

Etapa	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					

FIGURA 4.28 CAMINHOS DE MENOR CUSTO RESULTANTES E TABELA DE REPASSE PARA O NÓ U

Destino	Enlace
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

O roteamento inicial é mostrado na Figura 4.29(a) com os custos dos enlaces correspondentes à quantidade de tráfego transportada.

Quando o algoritmo LS é rodado de novo, o nó *y* determina — baseado nos custos dos enlaces mostrados na Figura 4.29(a) — que o caminho em sentido horário até *w* tem um custo de 1, ao passo que o caminho em sentido anti-horário até *w* (que estava sendo usado) tem o custo de $1 + e$. Por conseguinte, o caminho de menor custo de *y* até *w* é agora em sentido horário. De maneira semelhante, *x* determina que seu novo caminho de menor custo até *w* é também em sentido horário, resultando nos custos mostrados na Figura 4.29(b). Na próxima vez em que o algoritmo LS é rodado, os nós *x*, *y* e *z* detectam um caminho de custo zero até *w* na direção anti-horária, e todos dirigem seu tráfego para as rotas anti-horárias. Na próxima vez em que o algoritmo LS é rodado, os nós *x*, *y* e *z* então dirigem seu tráfego para as rotas em sentido horário.

O que pode ser feito para evitar essas oscilações (que podem ocorrer com qualquer algoritmo, e não apenas com um algoritmo LS, que use uma métrica de enlace baseada em congestionamento ou em atraso)? Uma solu-

FIGURA 4.29 OSCILAÇÕES COM ROTEAMENTO SENSÍVEL AO CONGESTIONAMENTO

a. Roteamento inicial

b. *x*, *y* detectam melhor caminho até *w* em sentido horário

c. *x*, *y*, *z* detectam melhor caminho até *w* em sentido anti-horário

d. *x*, *y*, *z*, detectam melhor caminho até *w* em sentido horário

ção seria tornar obrigatório que os custos dos enlaces não dependessem da quantidade de tráfego transportada — uma solução inaceitável, já que um dos objetivos do roteamento é evitar enlaces muito congestionados (por exemplo, enlaces com grande atraso). Outra solução seria assegurar que nem todos os roteadores rodassem o algoritmo LS ao mesmo tempo. Esta parece ser uma solução mais razoável, já que é de esperar que, mesmo que os roteadores rodem o algoritmo LS com idêntica periodicidade, o instante de execução do algoritmo não seja o mesmo em cada nó. O interessante é que os pesquisadores descobriram que os roteadores da Internet podem se autossincronizar [Floyd Synchronization, 1994]. Isto é, mesmo que inicialmente rodem o algoritmo com o mesmo período, mas em diferentes momentos, a instância de execução do algoritmo pode finalmente se tornar, e permanecer, sincronizada nos roteadores. Um modo de evitar essa autossincronização é cada roteador variar aleatoriamente o instante em que envia um anúncio de enlace.

Agora que examinamos o algoritmo de estado de enlace, vamos analisar outro importante algoritmo usado hoje na prática — o algoritmo de roteamento de vetor de distâncias.

4.5.2 O algoritmo de roteamento de vetor de distâncias (DV)

Enquanto o algoritmo LS usa informação global, o algoritmo de **vetor de distâncias** (***distance-vector* — DV**) é iterativo, assíncrono e distribuído. É *distribuído* porque cada nó recebe alguma informação de um ou mais vizinhos *diretamente ligados* a ele, realiza cálculos e, em seguida, distribui os resultados de seus cálculos para seus vizinhos. É *iterativo* porque esse processo continua até que mais nenhuma informação seja trocada entre vizinhos. (O interessante é que este é um algoritmo finito — não há nenhum sinal de que o cálculo deve parar; ele apenas para.) O algoritmo é *assíncrono* porque não requer que todos os nós rodem simultaneamente. Veremos que um algoritmo assíncrono, iterativo, finito e distribuído é muito mais interessante e divertido do que um algoritmo centralizado!

Antes de apresentar o algoritmo DV, é bom discutir uma relação importante que existe entre os custos dos caminhos de menor custo. Seja $d_x(y)$ o custo do caminho de menor custo do nó x ao nó y. Então, os menores custos estão relacionados segundo a famosa equação de Bellman-Ford:

$$d_x(y) = \min_v\{c(x,v) + d_v(y)\}, \qquad (4.1)$$

sendo o min_v da equação calculado para todos os vizinhos de x. A equação de Bellman-Ford é bastante intuitiva. Realmente, se após transitarmos de x para v tomarmos o caminho de menor custo de v a y, o custo do caminho será $c(x,v) + d_v(y)$. Como devemos começar viajando até algum vizinho v, o caminho de menor custo de x a y é o mínimo do conjunto dos $c(x,v) + d_v(y)$ calculados para todos os vizinhos v.

Mas, para aqueles que ainda se mostrem céticos quanto à validade da equação, vamos verificá-la para o nó de origem u e o nó de destino z na Figura 4.27. O nó de origem u tem três vizinhos: nós v, x e w. Percorrendo vários caminhos no grafo, é fácil ver que $d_v(z) = 5$, $d_x(z) = 3$ e $d_w(z) = 3$. Passando esses valores para a Equação 4.1, junto com os custos $c(u,v) = 2$, $c(u,x) = 1$ e $c(u,w) = 5$, temos $d_u(z) = \min\{2 + 5, 5 + 3, 1 + 3\} = 4$, que é, claro, verdade e que é, exatamente, o resultado conseguido com o algoritmo de Dijkstra para a mesma rede. Essa verificação rápida deve ajudá-lo a vencer qualquer ceticismo que ainda possa ter.

A equação de Bellman-Ford não é apenas uma curiosidade intelectual. Na verdade, ela tem uma importância prática significativa. Em particular, sua solução fornece os registros da tabela de repasse do nó x. Para verificar, seja v^* qualquer nó vizinho que represente o mínimo na Equação 4.1. Então, se o nó x quiser enviar um pacote ao nó y pelo caminho de menor custo, deverá, primeiro, repassá-lo para o nó v^*. Assim, a tabela de repasse do nó x especificaria o nó v^* como o roteador do próximo salto para o destino final y. Outra contribuição importante dessa equação é que ela sugere a forma da comunicação vizinho para vizinho que ocorrerá no algoritmo DV.

A ideia básica é a seguinte. Cada nó começa com $D_x(y)$, uma estimativa do custo do caminho de menor custo entre ele mesmo e o nó y, para todos os nós em N. Seja $\mathbf{D}_x = [D_x(y): y \text{ em } N]$ o vetor de distâncias do nó x, que é

o vetor de estimativas de custo de x até todos os outros nós, y, em N. Com o algoritmo DV cada nó x mantém os seguintes dados de roteamento:

- Para cada vizinho v, o custo $c(x,v)$ de x até o vizinho diretamente ligado a ele, v
- O vetor de distâncias do nó x, isto é, $\mathbf{D}_x = [D_x(y)$: y em $N]$, contendo a estimativa de x para seus custos até todos os destinos, y, em N
- Os vetores de distâncias de seus vizinhos, isto é, $\mathbf{D}_v = [D_v(y)$: y em $N]$ para cada vizinho v de x

No algoritmo distribuído, assíncrono, cada nó envia, a intervalos regulares, uma cópia do seu vetor de distâncias a cada um de seus vizinhos. Quando um nó x recebe um novo vetor de distâncias de qualquer de seus vizinhos v, ele armazena o vetor de distâncias de v e então usa a equação de Bellman-Ford para atualizar seu próprio vetor de distâncias, como a seguir:

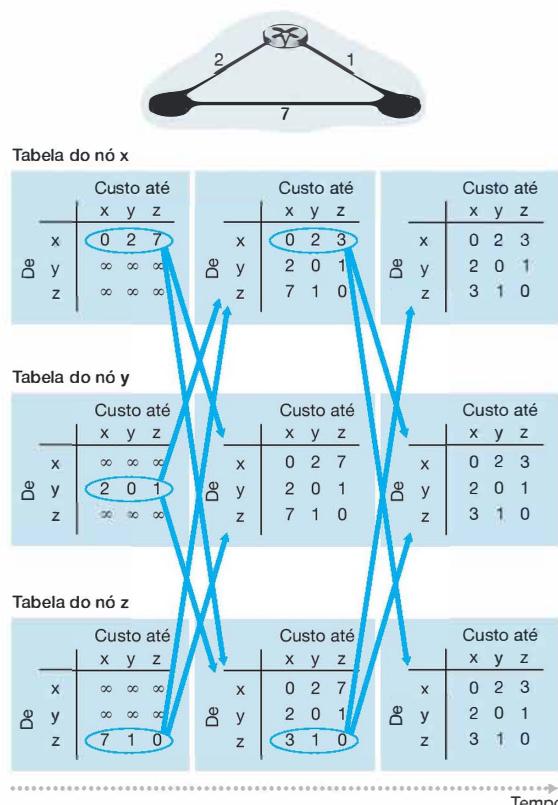
$$D_x(y) = \min_v\{c(x,v) + D_v(y)\} \qquad \text{para cada nó } y \text{ em } N$$

Se o vetor de distâncias do nó x tiver mudado como resultado dessa etapa de atualização, o nó x então enviará seu vetor de distâncias atualizado para cada um de seus vizinhos que, por sua vez, podem atualizar seus próprios vetores de distâncias. Parece milagre, mas, contanto que todos os nós continuem a trocar seus vetores de distâncias de forma assíncrona, cada estimativa de custo $D_x(y)$ convergirá para $d_x(y)$, que é, na verdade, o custo do caminho de menor custo do nó x ao nó y [Bertsekas, 1991]!

Algoritmo de vetor de distâncias (DV)

Para cada nó, x:

```
1  Inicialização:
2      para todos os destinos y em N:
3          Dx(y) = c(x,y) /* se y não é um vizinho então c(x,y) = ∞ */
4  para cada vizinho w
5      Dw(y) = ? para todos os destinos y em N
6  para cada vizinho w
7      envia vetor de distâncias Dx  = [Dx(y): y em N] para w
8
9  loop
10     espere (até que ocorra uma mudança no custo do enlace ao vizinho
11            w ou até a recepção de um vetor de distâncias do vizinho w)
12
13     para cada y em N:
14         Dx(y) = minv{c(x,v) + Dv(y)}
15
16     se Dx(y) mudou para algum destino y
17         envia vetor de distâncias Dx = [Dx(y): y em N] para todos os vizinhos
18
19 para sempre
```

No algoritmo DV, um nó x atualiza sua estimativa do vetor de distâncias quando percebe uma mudança de custo em um dos enlaces ligados diretamente a ele ou recebe uma atualização do vetor de distâncias de algum vizinho. Mas, para atualizar sua própria tabela de repasse para um dado destino y, o que o nó x de fato precisa saber não é a distância do caminho mais curto até y, mas qual nó vizinho $v^*(y)$ é o roteador do próximo salto ao longo do caminho mais curto até y. Como era de se esperar, o roteador do próximo salto $v^*(y)$ é o vizinho v que representar o mínimo na Linha 14 do algoritmo DV. (Se houver vários vizinhos v que representem o mínimo, então $v^*(y)$ pode ser qualquer um dos vizinhos minimizadores.) Assim, nas Linhas 13-14, para cada destino y, o nó x também determina $v^*(y)$ e atualiza sua tabela de repasse para o destino y.

Lembre-se de que o algoritmo LS é um algoritmo global no sentido de que requer que cada nó obtenha, primeiro, um mapa completo da rede antes de rodar o algoritmo de Dijkstra. O algoritmo DV é *descentralizado* e não usa essa informação global. De fato, a única informação que um nó terá são os custos dos enlaces até os

vizinhos diretamente ligados a ele e as informações que recebe desses vizinhos. Cada nó espera uma atualização de qualquer vizinho (Linhas 10–11), calcula seu novo vetor de distâncias ao receber uma atualização (Linha 14) e distribui seu novo vetor de distâncias a seus vizinhos (Linhas 16–17). Algoritmos semelhantes ao DV são utilizados em muitos protocolos de roteamento na prática, entre eles o RIP e o BGP da Internet, o ISO IDRP, o IPX da Novell, e o ARPAnet original.

A Figura 4.30 ilustra a operação do algoritmo DV para a rede simples de três nós mostrada na parte superior da figura. A operação do algoritmo é ilustrada de um modo síncrono, no qual todos os nós recebem vetores de distâncias simultaneamente de seus vizinhos, calculam seus novos vetores de distâncias e informam a seus vizinhos se esses vetores mudaram. Após estudar esse exemplo, você deve se convencer de que o algoritmo também opera corretamente em modo assíncrono, com cálculos de nós e atualizações de geração/recepção ocorrendo a qualquer instante.

A coluna mais à esquerda na figura mostra três **tabelas de roteamento** iniciais para cada um dos três nós. Por exemplo, a tabela no canto superior à esquerda é a tabela de roteamento inicial do nó *x*. Dentro de uma tabela de roteamento específica, cada linha é um vetor de distâncias — em especial, a tabela de roteamento de cada nó inclui seu próprio vetor de distâncias e os vetores de cada um de seus vizinhos. Assim, a primeira linha da tabela de roteamento inicial do nó *x* é $\mathbf{D}_x = [D_x(x), D_x(y), D_x(z)] = [0, 2, 7]$. A segunda linha e a terceira linha nessa tabela são os vetores de distâncias recebidos mais recentemente dos nós *y* e *z*. Como na inicialização o nó *x* não recebeu nada do nó *y* ou *z*, os registros da segunda linha e da terceira linha estão definidos como infinito.

FIGURA 4.30 ALGORITMO DE VETOR DE DISTÂNCIAS (DV)

2 1 7

Tabela do nó x

De \ Custo até	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

De \ Custo até	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

De \ Custo até	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

Tabela do nó y

De \ Custo até	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

De \ Custo até	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

De \ Custo até	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

Tabela do nó z

De \ Custo até	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

De \ Custo até	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

De \ Custo até	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

Tempo

Após a inicialização, cada nó envia seu vetor de distâncias a cada um de seus dois vizinhos. Isso é ilustrado na Figura 4.30 pelas setas que vão da primeira coluna de tabelas até a segunda. Por exemplo, o nó x envia seu vetor de distâncias $\mathbf{D}_x = [0, 2, 7]$ a ambos os nós y e z. Após receber as atualizações, cada nó recalcula seu próprio vetor de distâncias. Por exemplo, o nó x calcula

$$D_x(x) = 0$$
$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} = \min\{2 + 0, 7 + 1\} = 2$$
$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} = \min\{2 + 1, 7 + 0\} = 3$$

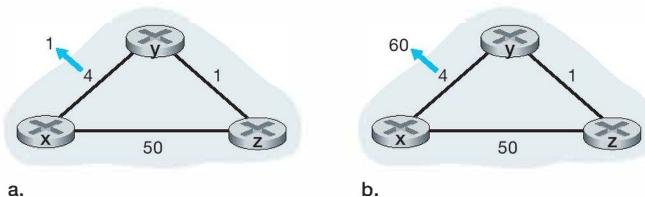
Por conseguinte, a segunda coluna mostra, para cada nó, o novo vetor de distâncias do nó, junto com os vetores de distâncias que acabou de receber de seus vizinhos. Observe, por exemplo, que a estimativa do nó x para o menor custo até o nó z, $D_x(z)$, mudou de 7 para 3. Note também que, para par ao nó x, o nó vizinho y alcança o mínimo na linha 14 do algoritmo DV; assim, nesse estágio do algoritmo, temos que, no nó x, $v^*(y) = y$ e $v^*(z) = y$.

Depois que recalculam seus vetores de distâncias, os nós enviam novamente seus vetores de distâncias recalculados a seus vizinhos (se houver uma mudança). Isso é ilustrado na Figura 4.30 pelas setas que vão da segunda até a terceira coluna de tabelas. Note que apenas os nós x e z enviam atualizações: o vetor de distâncias do nó y não mudou, então esse nó não envia uma atualização. Após recebê-las, os nós então recalculam seus vetores de distâncias e atualizam suas tabelas de roteamento, que são mostradas na terceira coluna.

O processo de receber vetores de distâncias atualizados de vizinhos, recalcular os registros de tabelas de roteamento e informar aos vizinhos os custos modificados do caminho de menor custo até o destino continua até que mais nenhuma mensagem de atualização seja enviada. Nesse ponto, não ocorrerá mais nenhum cálculo de tabela de roteamento e o algoritmo entra em estado de inatividade; isto é, todos os nós estarão realizando a espera nas Linhas 10-11 do algoritmo DV. Este permanece no estado de inatividade até que o custo de um enlace mude, como veremos a seguir.

Algoritmo de vetor de distâncias: mudanças no custo do enlace e falha no enlace

Quando um nó que está rodando o algoritmo DV detecta uma mudança no custo do enlace dele mesmo até um vizinho (Linhas 10-11), ele atualiza seu vetor de distâncias (Linhas 13-14) e, se houver uma modificação no custo do caminho de menor custo, informa a seus vizinhos (Linhas 16-17) seu novo vetor de distâncias. A Figura 4.31(a) ilustra um cenário em que o custo do enlace de y a x muda de 4 para 1. Destacamos aqui somente os registros na tabela de distâncias de y e z até o destino x. O algoritmo DV faz que ocorra a seguinte sequência de eventos:

- No tempo t_0, y detecta a mudança no custo do enlace (o custo mudou de 4 para 1), atualiza seu vetor de distâncias e informa essa mudança a seus vizinhos, já que o vetor mudou.
- No tempo t_1, z recebe a atualização de y e atualiza sua própria tabela. Calcula um novo menor custo para x (cujo custo diminuiu de 5 para 2), e envia seu novo vetor de distâncias a seus vizinhos.
- No tempo t_2, y recebe a atualização de z e atualiza sua tabela de distâncias. Os menores custos de y não mudaram e, por conseguinte, y não envia nenhuma mensagem a z. O algoritmo entra em estado de inatividade.

Assim, apenas duas iterações são necessárias para o algoritmo DV alcançar o estado de inatividade. A boa notícia sobre a redução do custo entre x e y se propagou rapidamente pela rede.

Agora vamos considerar o que pode acontecer quando o custo de um enlace *aumenta*. Suponha que o custo do enlace entre x e y aumente de 4 para 60, conforme mostra a Figura 4.31(b).

1. Antes da mudança do custo do enlace, $D_y(x) = 4$, $D_y(z) = 1$, $D_z(y) = 1$, e $D_z(x) = 5$. No tempo t_0, y detecta uma mudança no custo do enlace (o custo mudou de 4 para 60). y calcula seu novo caminho de custo mínimo até x, de modo a ter um custo de

$$D_y(x) = \min\{c(y,x) + D_x(x), c(y,z) + D_z(x)\} = \min\{60 + 0, 1 + 5\} = 6$$

FIGURA 4.31 MUDANÇAS NO CUSTO DO ENLACE

É claro que, com nossa visão global da rede, podemos ver que esse novo custo via z está *errado*. Mas as únicas informações que o nó y tem é que seu custo direto até x é 60 e que z disse a y que z pode chegar a x com um custo de 5. Assim, para chegar a x, y teria de fazer a rota através de z, com a expectativa de que z será capaz de chegar a x com um custo de 5. A partir de t_1, temos um ***loop* de roteamento** — para poder chegar a x, y faz a rota através de z, que por sua vez faz a rota através de y. Um *loop* de roteamento é como um buraco negro — um pacote destinado a x que ao chegar a y ou a z a partir do momento t_1 vai ricochetear entre esses dois nós para sempre (ou até que as tabelas de repasse sejam mudadas).

2. Tão logo o nó y tenha calculado um novo custo mínimo até x, ele informará a z esse novo vetor de distâncias no tempo t_1.
3. Algum tempo depois de t_1, z recebe o novo vetor de distâncias de y, que indica que o custo mínimo de y até x é 6. z sabe que pode chegar até y com um custo de 1 e, por conseguinte, calcula um novo menor custo até x, $D_2(x) = \min\{50 + 0{,}1 + 6\} = 7$. Uma vez que o custo mínimo de z até x aumentou, z informa a y o seu novo vetor de distâncias em t_2.
4. De maneira semelhante, após receber o novo vetor de distâncias de z, y determina $D_y(x) = 8$ e envia a z seu vetor de distâncias. Então z determina $D_z(x)= 9$ e envia a y seu vetor de distâncias e assim por diante.

Por quanto tempo esse processo continua? Pode ter certeza de que o *loop* persistirá por 44 iterações (trocas de mensagens entre y e z) até que z possa, enfim, calcular que o custo de seu caminho via y é maior do que 50. Nesse ponto, z (enfim!) determinará que seu caminho de menor custo até x é via sua conexão direta com x. Então y fará a rota até x via z. O resultado das más notícias sobre o aumento do custo do enlace na verdade viajou devagar! O que teria acontecido se o custo do enlace $c(y, x)$ tivesse mudado de 4 para 10.000 e o custo $c(z, x)$ fosse 9.999? Por causa de cenários como esses, o problema que acabamos de examinar é, às vezes, denominado problema de contagem ao infinito.

Algoritmo de vetor de distâncias: adição de reversão envenenada

O cenário específico de looping que acabamos de descrever pode ser evitado usando uma técnica denominada reversão envenenada (*poisoned reverse*). A ideia é simples — se a rota de z para chegar a x passa por y, então z deve anunciar a y que sua distância a x é infinita, isto é, z anunciará a y que $D_z(x) = \infty$ (mesmo que z saiba que, na verdade, $D_z(x) = 5$). z continuará contando essa mentirinha inocente a y enquanto a rota de z a x estiver passando por y. Enquanto y acreditar que z não tem nenhum caminho até x, y jamais tentará a rota até x por z, contanto que a rota de z a x continue a passar por y (e ele minta sobre isso).

Agora vamos ver como a reversão envenenada resolve o problema específico do *looping* que encontramos antes na Figura 4.31(b). Como resultado da reversão envenenada, a tabela de distâncias de y indica que $D_z(x) = \infty$. Quando o custo do enlace (x, y) muda de 4 para 60 no tempo t_0, y atualiza sua tabela e continua a estabelecer rotas diretamente para x, embora com um custo mais alto do que 60, e informa a z o seu novo custo até x, isto é, $D_y(x) = 60$. Após receber a atualização em t_1, z imediatamente desloca sua rota para x, para que passe pelo enlace direto (z, x) a um custo de 50. Como este é um novo menor custo até x, e já que o caminho não passa mais por y, z agora informa a y que $D_z(x) = 50$ em t_2. Após receber a atualização de z, y atualiza sua tabela de distâncias com

$D_z(x) = 51$. E, também, como z está agora no caminho de menor custo até x, y envenena o caminho inverso de z a x, informando a z, no tempo t_3, que $D_y(x) = \infty$ (mesmo que y saiba que, na verdade, $D_y(x) = 51$).

A reversão envenenada resolve o problema geral da contagem até o infinito? Não resolve. É bom que você se convença de que *loops* que envolvem três ou mais nós (e não apenas dois nós imediatamente vizinhos) não serão detectados pela técnica da reversão envenenada.

Uma comparação entre os algoritmos de roteamento de estado de enlace (LS) e de vetor de distâncias (DV)

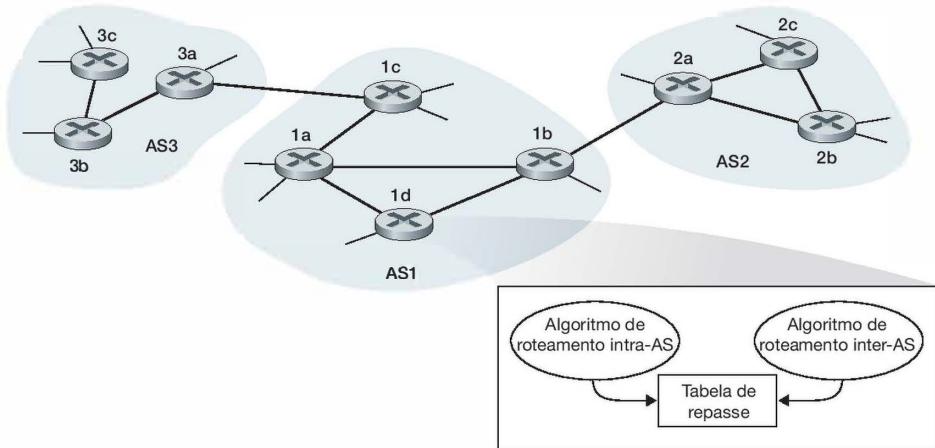
Os algoritmos DV e LS adotam abordagens complementares em relação ao cálculo do roteamento. No algoritmo DV, cada nó fala *somente* com os vizinhos diretamente conectados a ele, mas informa a esses vizinhos as estimativas de menor custo entre ele mesmo e *todos* os outros nós da rede (isto é, todos os que ele sabe que existem). No algoritmo LS, cada nó fala com *todos* os outros nós (por difusão), mas informa *somente* os custos dos enlaces diretamente ligados a ele. Vamos concluir nosso estudo sobre algoritmos de estado de enlace e de vetor de distâncias com uma rápida comparação de alguns de seus atributos. Lembre-se de que N é o conjunto de nós (roteadores) e E é o conjunto de arestas (enlaces).

- *Complexidade da mensagem*. Vimos que o LS requer que cada nó saiba o custo de cada enlace da rede. Isso exige que sejam enviadas $O(|N|\,|E|)$ mensagens. E, também, sempre que o custo de um enlace muda, o novo custo deve ser enviado a todos os nós. O algoritmo DV requer troca de mensagens entre vizinhos diretamente conectados a cada iteração. Já vimos que o tempo necessário para que o algoritmo convirja pode depender de muitos fatores. Quando o custo do enlace muda, o algoritmo DV propaga os resultados do custo modificado do enlace apenas se o novo custo resultar em mudança no caminho de menor custo para um dos nós ligado ao enlace.
- *Velocidade de convergência*. Já vimos que nossa implementação de LS é um algoritmo $O(|N|^2)$ que requer $O(N|\,|E|)$ mensagens. O algoritmo DV pode convergir lentamente e pode ter *loops* de roteamento enquanto estiver convergindo. O algoritmo DV também tem o problema da contagem até o infinito.
- *Robustez*. O que pode acontecer se um roteador falhar, se comportar mal ou for sabotado? Sob o LS, um roteador poderia transmitir um custo incorreto para um de seus enlaces diretos (mas não para outros). Um nó poderia também corromper ou descartar quaisquer pacotes recebidos como parte de uma difusão de estado de enlace. Mas um nó LS está calculando apenas suas próprias tabelas de roteamento; os outros nós estão realizando cálculos semelhantes para si próprios. Isso significa que, sob o LS, os cálculos de rota são, de certa forma, isolados, fornecendo um grau de robustez. Sob o DV, um nó pode anunciar incorretamente caminhos de menor custo para qualquer destino, ou para todos os destinos. (Na verdade, em 1997, um roteador que estava funcionando mal em um pequeno ISP forneceu aos roteadores nacionais de *backbone* tabelas de roteamento errôneas. Isso fez outros roteadores inundarem de tráfego o roteador que estava funcionando mal. Com isso, grandes porções da Internet ficaram desconectadas durante muitas horas [Neumann, 1997].) De modo geral, notamos que, a cada iteração, um cálculo de nó em DV é passado adiante a seu vizinho e, em seguida, indiretamente ao vizinho de seu vizinho na iteração seguinte. Nesse sentido, sob o DV, um cálculo incorreto do nó pode ser difundido pela rede inteira.

No final, nenhum algoritmo *ganha* do outro; na verdade, ambos são usados na Internet.

Outros algoritmos de roteamento

Os algoritmos LS e DV que estudamos não somente são utilizados em grande escala na prática, mas também são, na essência, os *únicos* utilizados hoje na Internet. Não obstante, muitos algoritmos de roteamento fo-

ram propostos por pesquisadores nos últimos 30 anos, abrangendo desde o extremamente simples até o muito sofisticado e complexo. Há uma grande classe de algoritmos de roteamento cuja base é considerar o tráfego como fluxos entre origens e destinos em uma rede. Nessa abordagem, o problema do roteamento pode ser formulado em termos matemáticos como um problema de otimização restrita, conhecido como problema de fluxo da rede [Bertsekas, 1991]. Ainda outro conjunto de algoritmos de roteamento que mencionamos aqui são os derivados do mundo da telefonia. Esses **algoritmos de roteamento de comutação de circuitos** são de interesse para redes de comutação de circuitos nos casos em que devem ser reservados recursos (por exemplo, buffers ou uma fração da largura de banda do enlace) por enlace para cada conexão roteada pelo enlace. Embora a formulação do problema do roteamento talvez pareça bem diferente da do problema do roteamento de menor custo que vimos neste capítulo, existem muitas semelhanças, ao menos quanto ao algoritmo de busca de caminhos (algoritmo de roteamento). Veja em Ash [1998]; Ross [1995]; Girard [1990] uma discussão detalhada dessa área de pesquisa.

4.5.3 Roteamento hierárquico

Quando estudamos os algoritmos LS e DV, consideramos a rede apenas como uma coleção de roteadores interconectados. Um roteador não se distinguia de outro no sentido de que todos rodavam o mesmo algoritmo de roteamento para calcular os caminhos de roteamento pela rede inteira. Na prática, esse modelo e sua visão de um conjunto homogêneo de roteadores, todos rodando o mesmo algoritmo de roteamento, é um tanto simplista por pelo menos duas razões importantes:

- *Escala.* À medida que aumenta o número de roteadores, a sobrecarga relativa ao cálculo, ao armazenamento e à comunicação de informações de roteamento (por exemplo, atualizações de estado de enlace ou mudanças no caminho de menor custo) se torna proibitiva. A Internet pública de hoje consiste em centenas de milhões de hospedeiros. Armazenar informações de roteamento para cada um desses hospedeiros evidentemente exigiria quantidades enormes de memória. Com a sobrecarga exigida para transmitir atualizações do estado de enlace por difusão entre todos os roteadores da Internet, não sobraria nenhuma largura de banda para enviar pacotes de dados! Um algoritmo DV que fizesse iterações entre esse número tão grande de roteadores decerto jamais convergiria! Fica claro que algo deve ser feito para reduzir a complexidade do cálculo de rotas em redes tão grandes como a Internet pública.
- *Autonomia administrativa.* Embora os pesquisadores tendam a ignorar questões como o desejo das empresas de controlar seus roteadores como bem entendem (por exemplo, rodar qualquer algoritmo de roteamento que escolherem) ou ocultar do público externo aspectos da organização interna das redes, essas considerações são importantes. Idealmente, uma organização deveria poder executar e administrar sua rede como bem entendesse, mantendo a capacidade de conectar sua rede a outras redes externas.

Esses problemas podem ser resolvidos agrupando roteadores em **sistemas autônomos** (***autonomous systems* — ASs**), com cada AS consistindo em um grupo de roteadores sob o mesmo controle administrativo (por exemplo, operados pelo mesmo ISP ou pertencentes a uma mesma rede corporativa). Todos os roteadores dentro do mesmo AS rodam o mesmo algoritmo de roteamento (por exemplo, LS ou DV) e dispõem das informações sobre cada um dos outros — exatamente como foi o caso do modelo idealizado da seção anterior. O algoritmo de roteamento que roda dentro de um AS é denominado um **protocolo de roteamento intrassistema autônomo**. É claro que será necessário conectar os ASs entre si e, assim, um ou mais dos roteadores em um AS terá a tarefa adicional de ficar responsável por transmitir pacotes a destinos que estão fora do AS — esses roteadores são denominados **roteadores de borda** (*gateway routers*).

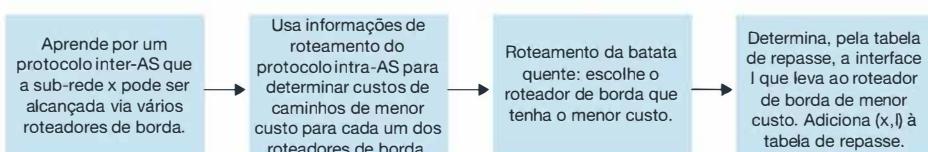
A Figura 4.32 ilustra um exemplo simples com três ASs: AS1, AS2 E AS3. Na figura, as linhas escuras representam conexões diretas de enlaces entre pares e roteadores. As linhas mais finas e interrompidas que saem dos roteadores representam sub-redes conectadas diretamente a eles. O AS1 tem quatro roteadores, 1a, 1b, 1c e 1d, e cada qual roda o protocolo de roteamento utilizado dentro do AS1. Assim, cada um desses quatro roteadores

FIGURA 4.32 UM EXEMPLO DE SISTEMAS AUTÔNOMOS INTERCONECTADOS

sabe como transmitir pacotes ao longo do caminho ideal para qualquer destino dentro de AS1. De maneira semelhante, cada um dos sistemas autônomos AS2 e AS3 tem três roteadores. Note que os protocolos de roteamento intra-AS que rodam em AS1, AS2 e AS3 não precisam ser os mesmos. Note também que os roteadores 1b, 1c, 2a e 3a são roteadores de borda.

Agora já deve estar claro como os roteadores em um AS determinam caminhos de roteamento para pares origem-destino internos ao AS. Mas ainda há uma peça faltando nesse quebra-cabeça de roteamento fim a fim. De que forma um roteador que está dentro de algum AS sabe como rotear um pacote até um destino que está fora do AS? Essa pergunta é fácil de responder se o AS tiver somente um roteador de borda que se conecta com somente outro AS. Nesse caso, como o algoritmo de roteamento intra-AS do sistema autônomo determinou o caminho de menor custo entre cada roteador interno e o de borda, cada roteador interno sabe como deve enviar o pacote. Ao recebê-lo, o roteador de borda o repassa para o único enlace que leva ao exterior do AS. Então, o AS que está na outra extremidade do enlace assume a responsabilidade de rotear o pacote até seu destino final. Como exemplo, suponha que o roteador 2b da Figura 4.32 receba um pacote cujo destino está fora do AS2. O roteador 2b, então, o transmite ao roteador 2a ou 2c, como especificado pela tabela de repasse de 2b, que foi configurada pelo protocolo de roteamento intra-AS de AS2. O pacote eventualmente chegará ao roteador de borda 2a, que o repassará ao 1b. Tão logo o pacote tenha saído de 2a, termina o trabalho de AS2 com referência a esse pacote.

Portanto, o problema é fácil quando o AS de origem tem apenas um enlace que leva para fora do AS. Mas, e se o AS de origem tiver dois ou mais enlaces (passando por um ou mais roteadores de borda) que levam para fora do AS? Então, o problema de saber para onde repassar o pacote torna-se bem mais desafiador. Por exemplo, considere um roteador em AS1 e suponha que ele recebe um pacote cujo destino está fora do AS. É claro que o roteador deveria repassar o pacote para um de seus dois roteadores de borda, 1b ou 1c, mas para qual deles? Para resolver esse problema, AS1 (1) precisa saber quais destinos podem ser alcançados via AS2 e quais podem ser alcançados via AS3 e (2) precisa propagar a informação a todos os roteadores dentro de AS1, de modo que cada roteador possa configurar sua tabela de repasse para manipular destinos externos ao AS. Essas duas tarefas — obter informações sobre as condições de alcance de ASs vizinhos e propagá-las a todos os outros roteadores internos ao AS — são gerenciadas pelo **protocolo de roteamento inter-AS.** Visto que tal protocolo envolve comunicação entre dois ASs, esses dois ASs comunicantes devem rodar o mesmo protocolo de roteamento inter-AS. De fato, na Internet, todos os ASs rodam o mesmo protocolo de roteamento inter-AS, denominado BGP4, que discutiremos na próxima seção. Como ilustrado na Figura 4.32, cada roteador recebe informações de um protocolo de roteamento intra-AS e de um protocolo de roteamento inter-AS, e usa as informações de ambos para configurar sua tabela de repasse.

Como exemplo, considere uma sub-rede *x* (identificada por seu endereço "ciderizado") e suponha que AS1 sabe, por meio do protocolo de roteamento inter-AS, que a sub-rede *x* pode ser alcançada de AS3, mas *não* pode ser alcançada de AS2. Então, AS1 propaga essa informação a todos os seus roteadores. Quando o roteador 1d fica sabendo que a sub-rede *x* pode ser alcançada de AS3 e, por conseguinte, do roteador de borda 1c, determina, com base na informação fornecida pelo protocolo de roteamento intra-AS, a interface de roteador que está no caminho de menor custo entre o roteador 1d e o roteador de borda 1c. Seja *I* essa interface. O roteador 1d então pode colocar o registro (*x*, *I*) em sua tabela de repasse. (Esse exemplo, e outros apresentados nesta seção, passam as ideias gerais, mas são simplificações do que de fato acontece na Internet. Na seção seguinte daremos uma descrição mais detalhada, se bem que mais complicada, quando discutirmos o BGP.)

Continuando com nosso exemplo anterior, suponha agora que AS2 e AS3 estão conectados com outros ASs, que não aparecem no diagrama. Suponha também que AS1 fica sabendo, pelo protocolo de roteamento inter-AS, que a sub-rede *x* pode ser alcançada a partir de AS2, via roteador de borda 1b, e também de AS3, via roteador de borda 1c. Então, AS1 propagaria essa informação a todos os seus roteadores, incluindo 1d. Para configurar sua tabela de repasse, o roteador 1d teria de determinar para qual roteador de borda, 1b ou 1c, deve dirigir pacotes destinados à sub-rede *x*. Uma abordagem, que costuma ser empregada na prática, é utilizar o **roteamento da batata quente**. Com ele, o AS se livra do pacote (a batata quente) o mais depressa possível (mais precisamente, com o menor custo possível). Isso é feito obrigando um roteador a enviar o pacote ao roteador de borda que tiver o menor custo roteador-roteador de borda entre todos os que têm um caminho para o destino. No contexto do exemplo que estamos examinando, o roteamento da batata quente, rodando em 1d, usaria informação recebida do protocolo de roteamento intra-AS para determinar os custos de caminho até 1b e 1c, e então escolheria o de menor custo. Uma vez escolhido esse caminho, o roteador 1d adiciona em sua tabela de repasse um registro para a sub-rede *x*. A Figura 4.33 resume as ações executadas no roteador 1d para adicionar o novo registro para *x* na tabela de repasse.

Quando um AS fica sabendo de um destino por meio de um AS vizinho, pode anunciar essa informação de roteamento a alguns outros ASs vizinhos. Por exemplo, suponha que AS1 fica sabendo, por AS2, que a sub-rede *x* pode ser alcançada via AS2. Então, AS1 diria a AS3 que *x* pode ser atingida via AS1. Desse modo, se AS3 precisar rotear um pacote destinado a *x*, repassaria o pacote a AS1 que, por sua vez, o repassaria para AS2. Como veremos quando discutirmos BGP, um AS tem bastante flexibilidade para decidir quais destinos anuncia aos ASs vizinhos. Esta é uma decisão *política*, que em geral depende mais de questões econômicas do que técnicas.

Lembre-se de que dissemos na Seção 1.5 que a Internet consiste em uma hierarquia de ISPs interconectados. Então, qual é a relação entre ISPs e ASs? Você talvez pense que os roteadores em um ISP e os enlaces que os interconectam constituem um único AS. Embora esse seja com frequência o caso, muitos ISPs dividem sua rede em vários ASs. Por exemplo, alguns ISPs de nível 1 utilizam um AS para toda a sua rede; outros a subdividem em dezenas de ASs interconectados.

Em resumo, os problemas de escala e de autoridade administrativa são resolvidos pela definição de sistemas autônomos. Dentro de um AS, todos os roteadores rodam o mesmo protocolo de roteamento intrassistema autônomo. Entre eles, os ASs rodam o mesmo protocolo de roteamento inter-AS. O problema de escala é resolvido porque um roteador intra-AS precisa saber apenas dos outros roteadores dentro do AS. O problema de autoridade administrativa é resolvido, já que uma organização pode rodar o protocolo de roteamento intra-AS

FIGURA 4.33 ETAPAS DA ADIÇÃO DE UM DESTINO FORA DO AS À TABELA DE REPASSE DE UM ROTEADOR

que quiser; todavia, cada par de ASs conectados precisa rodar o mesmo protocolo de roteamento inter-AS para trocar informações de alcançabilidade.

Na seção seguinte, examinaremos dois protocolos de roteamento intra-AS (RIP e OSPF) e o protocolo de roteamento inter-AS (BGP), que são usados na Internet de hoje. Esses estudos de casos dão um bom arremate ao nosso estudo de roteamento hierárquico.

4.6 ROTEAMENTO NA INTERNET

Agora que já estudamos endereçamento na Internet e o protocolo IP, vamos voltar nossa atenção aos protocolos de roteamento da Internet. A tarefa deles é determinar o caminho tomado por um datagrama entre a origem e o destino. Veremos que esses protocolos incorporam muitos dos princípios que aprendemos antes neste capítulo. As abordagens de estado de enlace e de vetor de distâncias estudadas nas Seções 4.5.1 e 4.5.2 e a ideia de um sistema autônomo considerada na Seção 4.5.3 são fundamentais para o modo como o roteamento é feito na Internet hoje.

Lembre-se de que, na Seção 4.5.3, vimos que um sistema autônomo (AS) é um conjunto de roteadores que estão sob o mesmo controle administrativo e técnico e que rodam, todos, o mesmo protocolo de roteamento entre eles. Cada AS, por sua vez, normalmente contém várias sub-redes (aqui, usamos o termo sub-rede no sentido preciso de endereçamento, como na Seção 4.4.2).

4.6.1 Roteamento intra-AS na Internet: RIP

Um protocolo de roteamento intra-AS é usado para determinar como é rodado o roteamento dentro de um sistema autônomo (AS). Esses protocolos são também conhecidos como **protocolos de roteadores internos** (**IGP** — ***interior gateway protocols***). Historicamente, dois protocolos de roteamento têm sido usados para roteamento dentro de um sistema autônomo na Internet: o **protocolo de informações de roteamento**, **RIP** (**Routing Information Protocol**) e o **OSPF** (**Open Shortest Path First**). Um protocolo muito relacionado com o OSPF é o **IS-IS** [RFC 1142; Perlman, 1999]. Primeiro, discutiremos o RIP e, em seguida, consideraremos o OSPF.

O RIP foi um dos primeiros protocolos de roteamento intra-AS da Internet, e seu uso é ainda muito disseminado. Sua origem e seu nome vêm da arquitetura XNS (Xerox Network Systems). A ampla disponibilização do RIP se deveu em grande parte à inclusão, em 1982, na versão do UNIX do Berkeley Software Distribution (BSD), que suportava TCP/IP. A versão 1 do RIP está definida no [RFC 1058] e a versão 2, compatível com a versão 1, no [RFC 2453].

O RIP é um protocolo de vetor de distâncias que funciona de um modo muito parecido com o protocolo DV idealizado que examinamos na Seção 4.5.2. A versão do RIP especificada no RFC 1058 usa contagem de saltos como métrica de custo, isto é, cada enlace tem um custo 1. Por simplicidade, no algoritmo DV da Seção 4.5.2 os custos foram definidos entre pares de roteadores. No RIP (e também no OSPF), na realidade, os custos são definidos desde um roteador de origem até uma sub-rede de destino. O RIP usa o termo *salto* (*hop*), que é o número de sub-redes percorridas no caminho mais curto entre o roteador de origem e uma sub-rede de destino, inclusive. A Figura 4.34 ilustra um AS com seis sub-redes de folha. A tabela da figura indica o número de saltos desde o roteador de origem A até todas as sub-redes folha.

O custo máximo de um caminho é limitado a 15, restringindo assim o uso do RIP a sistemas autônomos que têm menos de 15 saltos de diâmetro. Lembre-se de que, em protocolos DV, roteadores vizinhos trocam vetores de distância entre si. O vetor de distâncias para qualquer roteador é a estimativa atual das distâncias dos caminhos de menor custo entre aquele roteador e as sub-redes no AS. No RIP, atualizações de roteamento são trocadas entre vizinhos a cada 30 s mais ou menos, usando uma **mensagem de resposta RIP**. A mensagem de resposta enviada por um roteador ou um hospedeiro contém uma lista de até 25 sub-redes de destino dentro do AS, bem como as distâncias entre o remetente e cada uma delas. Mensagens de resposta também são conhecidas como **anúncios RIP**.

Vamos examinar um exemplo simples de como funcionam os anúncios RIP. Considere a parte de um AS mostrada na Figura 4.35. Nessa figura, as linhas que conectam os roteadores representam sub-redes. Apenas os roteadores (*A*, *B*, *C* e *D*) e as sub-redes (*w*, *x*, *y*, *z*) selecionados são rotulados. As linhas tracejadas indicam que o AS continua; portanto, esse sistema autônomo tem muito mais roteadores e enlaces do que os mostrados na figura.

Cada roteador mantém uma tabela RIP denominada **tabela de roteamento.** A tabela de roteamento de um roteador inclui o vetor de distâncias e a tabela de repasse desse roteador. A Figura 4.36 mostra a tabela do roteador *D*. Note que essa tabela tem três colunas. A primeira é para a sub-rede de destino, a segunda indica a identidade do roteador seguinte no caminho mais curto até a sub-rede de destino e a terceira indica o número de saltos (isto é, o número de sub-redes que têm de ser atravessadas, incluindo a rede de destino) para chegar à sub-rede de destino no caminho mais curto. Para esse exemplo, a tabela mostra que, para enviar um datagrama do roteador *D* até a sub-rede de destino *w*, o datagrama deve primeiro ser repassado ao roteador vizinho *A*; a tabela também mostra que a sub-rede de destino *w* está a dois saltos de distância no caminho mais curto. De modo semelhante, indica que a sub-rede *z* está a sete saltos de distância via roteador *B*. Em princípio, uma tabela de roteamento terá apenas uma linha para cada sub-rede no AS, embora a versão 2 do RIP permita a agregação de registros de sub-redes usando técnicas de agregação de rotas semelhantes àquelas que examinamos na Seção 4.4. A tabela na Figura 4.36 e as subsequentes estão apenas parcialmente completas.

Suponha agora que 30 s mais tarde o roteador *D* receba do roteador *A* o anúncio mostrado na Figura 4.37. Note que esse anúncio nada mais é do que informações da tabela de roteamento do roteador *A*! A informação indica, em particular, que a sub-rede *z* está a apenas quatro saltos do roteador *A*. Ao receber o anúncio, o roteador *D* o reúne (Figura 4.37) à tabela de roteamento antiga (Figura 4.36). Em particular, o roteador *D* fica sabendo que agora há um novo caminho pelo roteador *A* até a sub-rede *z* que é mais curto do que o caminho pelo roteador *B*. Assim, o roteador *D* atualiza sua tabela para levar em conta o mais curto dos caminhos mais curtos, conforme mostra a Figura 4.38. Você poderia perguntar como o caminho mais curto até a sub-rede *z* se tornou mais curto ainda? Possivelmente, o algoritmo de vetor de distâncias descentralizado ainda estava em processo de convergên-

FIGURA 4.34 NÚMERO DE SALTOS DO ROTEADOR DE ORIGEM *A* ATÉ VÁRIAS SUB-REDES

Destino	Saltos
u	1
v	2
w	2
x	3
y	3
z	2

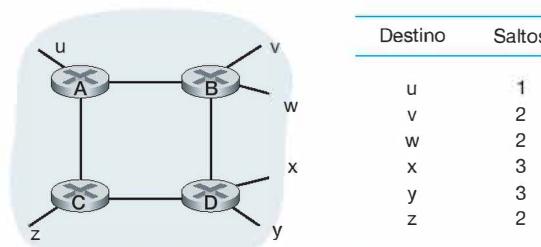



FIGURA 4.35 UMA PARTE DE UM SISTEMA AUTÔNOMO

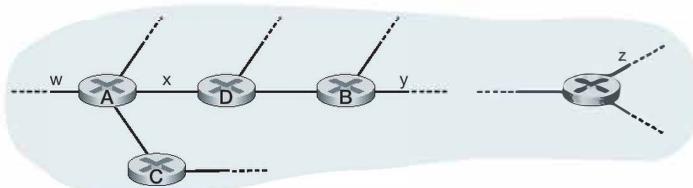
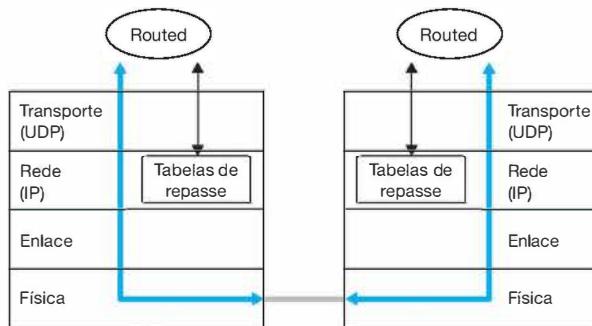



FIGURA 4.36 TABELA DE ROTEAMENTO NO ROTEADOR *D* ANTES DE RECEBER ANÚNCIO DO ROTEADOR *A*

Sub-rede de destino	Roteador seguinte	Número de saltos até o destino
w	A	2
y	B	2
z	B	7
x	—	1
....	....	....

FIGURA 4.37 ANÚNCIO VINDO DO ROTEADOR *A*

Sub-rede de destino	Roteador seguinte	Número de saltos até o destino
z	C	4
w	—	1
x	—	1
....	....	....

cia (veja a Seção 4.5.2) ou, talvez, novos enlaces e/ou roteadores tenham sido adicionados ao AS, mudando assim os caminhos mais curtos dentro dele.

Vamos agora considerar alguns dos aspectos da implementação do RIP. Lembre-se de que os roteadores RIP trocam anúncios a cada 30 s aproximadamente. Se um roteador não ouvir nada de seu vizinho ao menos uma vez a cada 180 s, esse vizinho será considerado impossível de ser alcançado dali em diante, isto é, está inoperante ou o enlace de conexão caiu. Quando isso acontece, o RIP modifica a tabela de roteamento local e, em seguida, propaga essa informação enviando anúncios a seus roteadores vizinhos (os que ainda podem ser alcançados). Um roteador pode também requisitar informação sobre o custo de seu vizinho até um dado destino usando uma mensagem de requisição RIP. Roteadores enviam mensagens de requisição e de resposta RIP uns aos outros usando o número de porta 520. O segmento UDP é carregado entre roteadores dentro de um datagrama IP padrão. O fato de o RIP usar um protocolo de camada de transporte (UDP) sobre um protocolo de camada de rede (IP) para implementar funcionalidade de camada de rede (um algoritmo de roteamento) pode parecer bastante confuso (e é!). Um exame mais profundo sobre como o RIP é implementado esclarecerá esse assunto.

A Figura 4.39 ilustra esquematicamente como o RIP costuma ser executado em um sistema UNIX, por exemplo, uma estação de trabalho UNIX que está servindo como um roteador. Um processo denominado *routed* (pronunciado como "*route dee*") roda o RIP, isto é, mantém informações de roteamento e troca mensagens com processos *routed* que rodam em roteadores vizinhos. Como o RIP é realizado como um processo da camada de aplicação (se bem que um processo muito especial, capaz de manipular as tabelas de roteamento dentro do núcleo do UNIX), ele pode enviar e receber mensagens por uma porta padrão e usar um protocolo de transporte padrão. Assim, o RIP é um protocolo de camada de aplicação (veja o Capítulo 2) que roda sobre UDP. Se estiver interessado em examinar uma implementação do RIP (ou dos protocolos OSPF e BGP, que estudaremos em seguida), consulte Quagga [2012].

FIGURA 4.38 TABELA DE ROTEAMENTO NO ROTEADOR *D* APÓS TER RECEBIDO ANÚNCIO DO ROTEADOR *A*

Sub-rede de destino	Roteador seguinte	Número de saltos até o destino
w	A	2
y	B	2
z	A	5
....	....	....

FIGURA 4.39 IMPLEMENTAÇÃO DO RIP COMO UM *DAEMON ROUTED*

Routed | Routed
Transporte (UDP) | Rede (IP) | Tabelas de repasse | Enlace | Física

4.6.2 Roteamento intra-AS na Internet: OSPF

Como o RIP, o roteamento OSPF é bastante usado para roteamento intra-AS na Internet. O OSPF e seu primo, IS-IS, muito parecido com ele, são em geral disponibilizados em ISPs de níveis mais altos, ao passo que o RIP está disponível em ISPs de níveis mais baixos e redes corporativas. O "*open*" do OSPF significa que as especificações do protocolo de roteamento estão abertas ao público (ao contrário do protocolo EIGRP da Cisco, por exemplo). A versão mais recente do OSPF, versão 2, está definida no RFC 2328, um documento público.

O OSPF foi concebido como sucessor do RIP e como tal tem uma série de características avançadas. Em seu âmago, contudo, é um protocolo de estado de enlace que usa inundação de informação de estado de enlace e um algoritmo de caminho de menor custo de Dijkstra. Com o OSPF, um roteador constrói um mapa topológico completo (isto é, um grafo) de todo o sistema autônomo. O roteador então roda localmente o algoritmo do caminho mais curto de Dijkstra para determinar uma árvore de caminho mais curto para todas as *sub-redes*, sendo ele próprio o nó raiz. Os custos de enlaces individuais são configurados pelo administrador da rede (veja "Princípios na prática: Configurando os pesos de enlaces no OSPF"). O administrador pode optar por estabelecer todos os custos de enlace em 1, conseguindo assim o roteamento com o mínimo de saltos, ou por designar para os enlaces pesos inversamente proporcionais à capacidade do enlace, de modo a desencorajar o tráfego a usar enlaces de largura de banda baixa. O OSPF não impõe uma política para o modo como são determinados os pesos dos enlaces (essa tarefa é do administrador da rede); em vez disso, oferece os mecanismos (protocolo) para determinar o caminho de roteamento de menor custo para um dado conjunto de pesos de enlaces.

Com OSPF, um roteador transmite por difusão informações de roteamento a *todos* os outros roteadores no sistema autônomo, não apenas a seus vizinhos. Um roteador transmite informações de estado de enlace por difusão sempre que houver uma mudança no estado de um enlace (por exemplo, uma mudança de custo ou uma mudança de estado para cima/para baixo). Também transmite o estado de um enlace periodicamente (pelo menos a cada 30 min), mesmo que não tenha havido mudança. O RFC 2328 observa que "essa atualização periódica de anúncios de enlace adiciona robustez ao algoritmo de estado de enlace". Anúncios OSPF são contidos em mensagens OSPF carregadas diretamente por IP, com um código protocolo de camada superior 89 para OSPF. Assim, o próprio protocolo OSPF tem de executar funcionalidades como transferência confiável de mensagem e transmissão de estado de enlace por difusão. O protocolo OSPF também verifica se os enlaces estão operacionais (via uma mensagem HELLO enviada a um vizinho ligado ao enlace) e permite que um roteador OSPF obtenha o banco de dados de um roteador vizinho referente ao estado do enlace no âmbito da rede.

Alguns dos avanços incorporados ao OSPF são:

- *Segurança*. Trocas entre roteadores OSPF (por exemplo, atualizações do estado de enlace) podem ser autenticadas. A autenticação garante que apenas roteadores de confiança conseguem participar do protocolo OSPF dentro de um AS, evitando, assim, que intrusos mal-intencionados (ou estudantes de rede

testando, por brincadeira, seu conhecimento recém-adquirido) injetem informações incorretas em tabelas de roteamento. Como padrão, pacotes OSPF entre roteadores não são autenticados e poderiam ser forjados. Dois tipos de autenticação podem ser configurados — simples e MD5 (veja o Capítulo 8 para obter uma discussão sobre MD5 e autenticação em geral). Com autenticação simples, a mesma senha é configurada em cada roteador. Quando um roteador envia um pacote OSPF, inclui a senha em texto claro (não criptografado). Logicamente, a autenticação simples não é segura. A autenticação MD5 é baseada em chaves secretas compartilhadas que são configuradas em todos os roteadores. Para cada pacote OSPF enviado, o roteador calcula o *hash* MD5 do conteúdo do pacote adicionado com a chave secreta. (Consulte a discussão sobre códigos de autenticação de mensagem no Capítulo 7.) Então, o roteador inclui no pacote OSPF o valor de *hash* resultante. O roteador receptor, usando a chave secreta pré-configurada, calculará um *hash* MD5 do pacote e o comparará com o valor de *hash* que este transporta, verificando assim sua autenticidade. Números de sequência também são utilizados com autenticação MD5 para proteção contra ataques por reenvio.

- *Caminhos múltiplos com o mesmo custo*. Quando vários caminhos até o destino têm o mesmo custo, o OSPF permite que sejam usados diversos (isto é, não é preciso escolher um único para carregar todo o tráfego quando existem vários de igual custo).
- *Suporte integrado para roteamento individual e em grupo (unicast e multicast)*. O *multicast* OSPF (MOSPF) [RFC 1584] fornece extensões simples ao OSPF para prover roteamento em grupo (um tópico que examinaremos com mais profundidade na Seção 4.7.2). O MOSPF usa o banco de dados de enlaces existente no OSPF e acrescenta um novo tipo de anúncio de estado de enlace ao mecanismo OSPF de transmissão de estado de enlace por difusão.

PRINCÍPIOS NA PRÁTICA

Configurando os pesos de enlaces no OSPF

Nossa discussão de roteamento de estado de enlace admitiu implicitamente que os pesos dos enlaces são determinados, que um algoritmo de roteamento como o OSPF é rodado e que o tráfego flui de acordo com as tabelas calculadas pelo algoritmo LS. Em termos de causa e efeito, os pesos dos enlaces são dados (isto é, vêm em primeiro lugar) e resultam (via algoritmo de Dijkstra) em caminhos de roteamento que minimizam o custo geral. Desse ponto de vista, pesos de enlaces refletem o custo da utilização de um enlace (por exemplo, se os pesos forem inversamente proporcionais à capacidade, então a utilização de enlaces de alta capacidade teria pesos menores e, assim, seriam mais atraentes do ponto de vista de roteamento) e o algoritmo de Dijkstra serve para minimizar o custo geral.

Na prática, a relação causa e efeito entre pesos de enlaces e caminhos de roteamento pode ser invertida — operadores de rede configuram pesos de enlaces de modo a obter caminhos de roteamento que cumpram certas metas de engenharia de tráfego [Fortz, 2000; Fortz, 2002]. Por exemplo, suponha que um operador de rede tenha uma estimativa de fluxo do tráfego que entra na rede em cada ponto de ingresso e destinado a cada ponto de saída da rede. Então, o operador poderia querer instituir um roteamento de tráfego do ponto de ingresso até o ponto de saída que minimizasse a utilização máxima em todos os enlaces da rede. Porém, com um algoritmo de roteamento como o OSPF, os principais controles de que o operador dispõe para ajustar o roteamento de fluxos pela rede são os pesos dos enlaces. Assim, para cumprir a meta de minimizar a utilização máxima do enlace, o operador tem de descobrir o conjunto de pesos de enlaces que alcance esse objetivo. Essa é uma inversão da relação causa e efeito — o roteamento de fluxos desejado é conhecido e os pesos dos enlaces OSPF têm de ser encontrados de um modo tal que o algoritmo de roteamento OSPF resulte nesse roteamento de fluxos desejado.

- *Suporte para hierarquia dentro de um único domínio de roteamento.* Talvez o avanço mais significativo do OSPF seja a capacidade de estruturar em hierarquia um sistema autônomo. Na Seção 4.5.3, vimos as muitas vantagens de estruturas de roteamento hierárquicas. Examinaremos a execução do roteamento OSPF hierárquico no restante desta seção.

Um sistema autônomo OSPF pode ser configurado hierarquicamente em áreas. Cada área roda seu próprio algoritmo de roteamento de estado de enlace OSPF, e cada roteador em uma área transmite seu estado de enlace, por difusão, a todos os outros roteadores daquela área. Dentro de cada área, um ou mais **roteadores de borda de área** são responsáveis pelo roteamento de pacotes fora da área. Por fim, exatamente uma área OSPF no AS é configurada para ser a área de ***backbone***. O papel primordial da área de *backbone* é rotear tráfego entre as outras áreas do AS. O *backbone* sempre contém todos os roteadores de borda de área que estão dentro do AS e pode conter também roteadores que não são de borda. O roteamento interárea dentro do AS requer que o pacote seja roteado primeiro até um roteador de borda de área (roteamento intra-área), em seguida roteado por meio do *backbone* até o roteador de borda de área que está na área de destino e, então, roteado até seu destino final.

O OSPF é um protocolo bastante complexo e, aqui, nosso tratamento teve de ser breve; Huitema [1998]; Moy [1998] e RFC 2328 oferecem detalhes adicionais.

4.6.3 Roteamento inter-AS: BGP

Acabamos de aprender como ISPs utilizam RIP e OSPF para determinar caminhos ótimos para pares origem-destino internos ao mesmo AS. Agora vamos examinar como são determinados caminhos para pares origem-destino que abrangem vários ASs. A versão 4 do **protocolo de roteador de borda (Border Gateway Protocol — BGP),** especificada no RFC 4271 (veja também [RFC 4274]), é o padrão, na prática, para roteamento entre sistemas autônomos na Internet de hoje. Esse protocolo é em geral denominado BGP4 ou apenas **BGP**. Na qualidade de um protocolo de roteamento inter-ASs (veja Seção 4.5.3), o BGP oferece a cada AS meios de:

1. Obter de ASs vizinhos informações de alcançabilidade de sub-redes.
2. Propagar a informação de alcançabilidade a todos os roteadores internos ao AS.
3. Determinar rotas "boas" para sub-redes com base na informação de alcançabilidade e na política do AS.

O BGP, sobretudo, permite que cada sub-rede anuncie sua existência ao restante da Internet. Uma sub-rede grita "Eu existo e estou aqui" e o BGP garante que todos os ASs da Internet saibam de sua existência e como chegar até ela. Não fosse o BGP, cada sub-rede ficaria isolada — sozinha e desconhecida pelo restante da Internet.

O básico do BGP

O BGP é de altíssima complexidade; livros inteiros foram dedicados ao assunto e muitas questões ainda não estão claras [Yannuzzi, 2005]. Além disso, mesmo após ler os livros e os RFCs, ainda assim você talvez ache difícil dominar o BGP por completo sem ter trabalhado com ele na prática durante muitos meses (se não anos) como projetista ou administrador de um ISP de nível superior. Não obstante, como o BGP é um protocolo absolutamente crítico para a Internet — na essência, é o que agrega tudo —, precisamos ao menos entender os aspectos básicos do seu funcionamento. Começamos descrevendo como o BGP poderia funcionar no contexto do exemplo de rede simples que já estudamos na Figura 4.32. Nesta descrição, baseamo-nos em nossa discussão sobre roteamento hierárquico na Seção 4.5.3; aconselhamos que você leia outra vez esse material.

No BGP, pares de roteadores trocam informações de roteamento por conexões TCP semipermanentes usando a porta 179. Essas conexões da Figura 4.32 são mostradas na Figura 4.40. Normalmente há uma conexão BGP TCP para cada enlace que liga diretamente dois roteadores que estão em dois ASs diferentes; assim, na Figura 4.40 há uma conexão TCP entre os roteadores de borda 3a e 1c e outra entre os roteadores de borda 1b e 2a. Tam-

bém há conexões BGP TCP semipermanentes entre roteadores dentro de um AS. Em particular, a Figura 4.40 apresenta uma configuração comum de uma conexão TCP para cada par de roteadores internos a um AS, criando uma malha de conexões TCP em cada AS. Os dois roteadores nas extremidades da conexão são denominados **pares BGP**, e a conexão TCP, junto com todas as mensagens BGP enviadas pela conexão, é denominada **sessão BGP**. Além disso, uma sessão BGP que abranja dois ASs é denominada **sessão BGP externa (eBGP)** e uma sessão BGP entre roteadores no mesmo AS é chamada uma **sessão BGP interna (iBGP)**. Na Figura 4.40, as sessões eBGP são indicadas pelas linhas de traços longos; as sessões iBGP são representadas pelas linhas de traços curtos. Note que as linhas das sessões BGP na Figura 4.40 nem sempre correspondem aos enlaces físicos na Figura 4.32.

O BGP permite que cada AS conheça quais destinos podem ser alcançados por meio de seus ASs vizinhos. No BGP, os destinos não são hospedeiros, mas **prefixos** *ciderizados*, e cada prefixo representa uma sub-rede ou um conjunto delas. Assim, por exemplo, suponha que haja quatro sub-redes conectadas ao AS2: 138.16.64/24, 138.16.65/24, 138.16.66/24 e 138.16.67/24. O AS2 então poderia agregar os prefixos dessas quatro sub-redes e utilizar o BGP para anunciar ao AS1 o prefixo único 138.16.64/22. Como outro exemplo, suponha que apenas as primeiras três estão em AS2 e que a quarta sub-rede, 138.16.67/24, está em AS3. Então, como descrito no quadro Princípios na Prática na Seção 4.4.2, como os roteadores usam combinação de prefixo mais longo para repassar datagramas, o AS3 poderia anunciar a AS1 o prefixo mais específico 138.16.67/24 e o AS2 *ainda* poderia anunciar ao AS1 o prefixo agregado 138.16.64/22.

Agora vamos examinar como o BGP distribuiria informações sobre a alcançabilidade de prefixos pelas sessões BGP mostradas na Figura 4.40. Como é de se esperar, usando a sessão eBGP entre os roteadores de borda 3a e 1c, AS3 envia a AS1 a lista de prefixos que podem ser alcançados a partir de AS3; e AS1 envia a AS3 a lista de prefixos que podem ser alcançados a partir de AS1. De modo semelhante, AS1 e AS2 trocam informações de alcançabilidade de prefixos por meio de seus roteadores de borda 1b e 2a. E, como também era de esperar, quando um roteador de borda (em qualquer AS) recebe prefixos conhecidos pelo BGP, ele usa suas sessões iBGP para distribuí-los aos outros roteadores no AS. Assim, todos os roteadores no AS1 se informarão sobre os prefixos de AS3, incluindo o roteador de borda 1b. O roteador de borda 1b (em AS1) pode reanunciar a AS2 os prefixos de AS3. Quando um roteador (seja ou não de borda) fica sabendo de um novo prefixo, cria um registro para ele em sua tabela de repasse, como descrito na Seção 4.5.3.

PRINCÍPIOS NA PRÁTICA

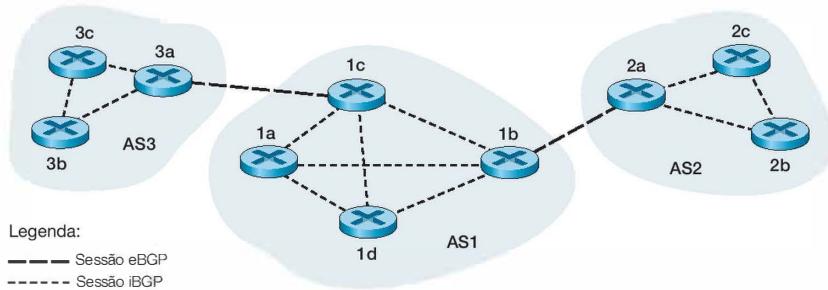
Obtendo presença na Internet: juntando o quebra-cabeça

Suponha que você tenha acabado de criar uma pequena rede com diversos servidores, incluindo um servidor Web público, que descreve os produtos e serviços da sua empresa, um de correio, do qual seus funcionários obtêm suas mensagens de correio eletrônico, e um de DNS. Claro, você gostaria que o mundo inteiro pudesse navegar em seu site para descobrir seus incríveis produtos e serviços. Além do mais, gostaria que seus funcionários pudessem enviar e receber correio eletrônico para clientes em potencial no mundo inteiro.

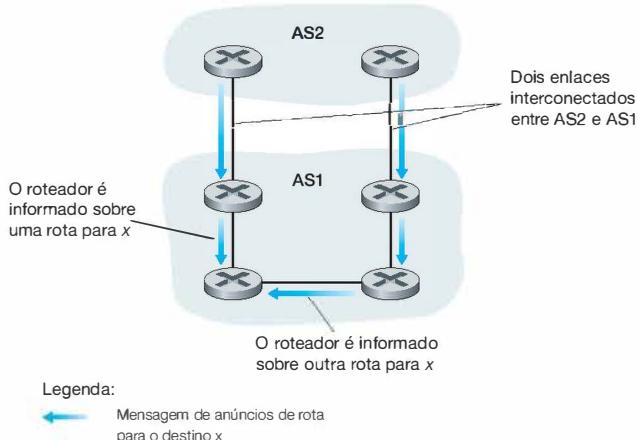
Para tender a esses objetivos, primeiro você precisa obter conectividade com a Internet, o que é feito contratando e conectando-se a um ISP local. Sua empresa terá um roteador de borda, que estará conectado a um roteador no seu ISP local. Essa conexão poderia ser uma DSL pela infraestrutura telefônica, uma linha privada com o roteador do ISP, ou uma das muitas outras soluções de acesso descritas no Capítulo 1. Seu ISP local também lhe oferecerá uma faixa de endereços IP, por exemplo, uma faixa de endereços /24 consistindo em 245 endereços. Ao obter sua conectividade física e sua faixa de endereços IP, você designará um dos seus endereços IP (na sua faixa de endereços) para o seu servidor Web, um para o seu servidor de correio, um para o seu servidor DNS, um para o seu roteador de borda e outros endereços IP para outros servidores e dispositivos na rede da sua empresa.

Além de contratar um ISP, você também precisará contratar um registrador da Internet para obter um nome de domínio para a sua empresa, conforme descrevemos no Capítulo 2. Por exemplo, se a sua empresa tiver o nome Xanadu Inc., você decerto tentará obter o nome de domínio xanadu.com, por exemplo.

Sua empresa também deverá obter presença no sistema DNS. Especificamente, como as pessoas de fora desejarão entrar em contato com seu servidor DNS para obter os endereços IP dos seus servidores, você também precisará oferecer ao seu registrador o endereço IP do seu servidor DNS. Seu registrador, então, colocará um registro para o seu servidor DNS (nome de domínio e endereço IP correspondente) nos servidores do domínio .com de nível superior, conforme descrevemos no Capítulo 2. Concluída essa etapa, qualquer usuário que saiba seu nome de domínio (por exemplo, xanadu.com) poderá obter o endereço IP do seu servidor DNS por meio do sistema DNS.

Para que as pessoas consigam descobrir os endereços IP do seu servidor Web, você terá de incluir nele registros que mapeiem o nome de hospedeiro do servidor (por exemplo, www.xanadu.com) ao endereço IP. Você desejará ter registros semelhantes para outros servidores publicamente disponíveis em sua empresa, incluindo o de correio. Dessa forma, se Alice quiser navegar pelo seu servidor Web, o sistema DNS entrará em contato com seu servidor DNS, achará o endereço IP do seu servidor Web e o dará a Alice. Assim, ela poderá estabelecer uma conexão TCP diretamente com o seu servidor Web.

Todavia, ainda resta uma etapa necessária e decisiva para permitir que outros do mundo inteiro acessem seu servidor Web. Considere o que acontece quando Alice, que conhece o endereço IP do seu servidor Web, envia um datagrama IP (por exemplo, um segmento TCP SYN) a esse endereço IP. Esse datagrama será direcionado pela Internet, visitando uma série de roteadores em muitos ASs diferentes, para enfim alcançar seu servidor Web. Quando qualquer um dos roteadores recebe o datagrama, ele procura um registro em sua tabela de repasse para determinar em qual porta de saída deverá encaminhá-lo. Portanto, cada roteador precisa saber a respeito do prefixo /24 da sua empresa (ou de algum registro agregado). Como um roteador pode saber o prefixo da sua empresa? Como já vimos, isso é feito por meio do BGP! Especificamente, quando sua empresa contrata um ISP local e recebe um prefixo (ou seja, uma faixa de endereços), seu ISP local usará o BGP para anunciar esse prefixo aos ISPs aos quais se conecta. Tais ISPs, por sua vez, usarão o BGP para propagar o anúncio. Por fim, todos os roteadores da Internet saberão a respeito do seu prefixo (ou sobre algum agregado que o inclua) e, desse modo, poderão repassar datagramas destinados a seus servidores Web e de correio de forma apropriada.

Atributos de caminho e rotas BGP

Agora que já temos um conhecimento preliminar do BGP, vamos um pouco mais fundo (e, mesmo assim, estaremos varrendo para baixo do tapete alguns dos detalhes menos importantes!). No BGP, um sistema autônomo é identificado por seu **número de sistema autônomo** (ASN) globalmente exclusivo [RFC 1930]. (Tecnicamente, nem todo AS tem um ASN. Em particular, um *stub* AS que carregue somente tráfego do qual é uma origem ou um destino em geral não terá um ASN; ignoramos essa tecnicidade em nossa discussão para que os detalhes não nos impeçam de observar melhor o quadro geral.) Números de ASs, como endereços IP, são designados por entidades regionais ICANN de registro [ICANN, 2012].

Quando um roteador anuncia um prefixo para uma sessão BGP, inclui vários **atributos BGP** juntamente com o prefixo. Na terminologia do BGP, um prefixo, junto com seus atributos, é denominado uma **rota**. Assim, pares BGP anunciam rotas uns aos outros. Dois dos atributos mais importantes são AS-PATH e NEXT-HOP:

- *AS-PATH*. Esse atributo contém os ASs pelos quais passou o anúncio para o prefixo. Quando um prefixo é passado para dentro de um AS, este adiciona seu ASN ao atributo AS-PATH. Por exemplo, considere a Figura 4.40 e suponha que o prefixo 138.16.64/24 seja anunciado primeiro de AS2 para AS1; se AS1 então anunciar o prefixo a AS3, o AS-PATH seria AS2 AS1. Roteadores usam o atributo AS-PATH para detectar e evitar *looping* de anúncios; em especial, se um roteador perceber que seu AS está contido na lista de caminhos, rejeitará o anúncio. Como discutiremos em breve, roteadores também usam o atributo AS-PATH ao escolher entre vários caminhos para o mesmo prefixo.
- Fornecendo o enlace crítico entre os protocolos de roteamento inter-AS e intra-AS, o atributo NEXT-HOP possui uma sutil mas importante utilidade. *O NEXT-HOP é a interface do roteador que inicia o AS-PATH.*

FIGURA 4.40 SESSÕES eBGP E iBGP

Para compreender mais esse atributo, vamos, de novo, nos referir à Figura 4.40. Considere o que acontece quando o roteador de borda 3a em AS3 anuncia uma rota a um roteador de borda 1c em AS1, utilizando eBGP. A rota inclui o prefixo anunciado, que chamaremos de *x*, e um AS-PATH para o prefixo. O anúncio também inclui o NEXT-HOP, endereço IP da interface 3a do roteador que conduz a 1c. (Lembre-se de que um roteador possui diversos endereços IP, um para cada uma das interfaces.) Considere agora o que ocorre quando o roteador 1d é informado sobre essa rota pelo iBGP. Após descobrir essa rota para *x*, o roteador 1d pode querer encaminhar pacotes para *x* por ela, ou seja, o roteador 1d pode querer incluir o registro (*x*, *l*) em sua tabela de repasse, na qual *l* representa sua interface que inicia o caminho de menor custo partindo de 1d em direção ao roteador de borda 1c. Para determinar *l*, 1d fornece o endereço IP no atributo NEXT--HOP a seu módulo de roteamento intra-AS. Observe que o algoritmo de roteamento intra-AS determinou o enlace de menor custo entre 1c e 3a. Desse caminho de menor custo de 1d à sub-rede 1c-3a, 1d determina sua interface do roteador *l* que inicia esse caminho e, então, adiciona a entrada (*x*, *l*) à sua tabela de repasse. Até que enfim! Em resumo, o atributo NEXT-HOP é usado por roteadores para configurar suas tabelas de repasse adequadamente.

- A Figura 4.41 ilustra outra situação em que o NEXT-HOP é necessário. Na figura, o AS1 e o AS2 estão conectados por dois enlaces interconectados. Um roteador em AS1 poderia descobrir duas rotas diferentes

FIGURA 4.41 ATRIBUTOS NEXT-HOP EM ANÚNCIOS SERVEM PARA DETERMINAR QUAL ENLACE INTERCONECTADO UTILIZAR

PRINCÍPIOS NA PRÁTICA

Juntando as partes: como um registro entra na tabela de repasse de um roteador?

Lembre-se de que um registro na tabela de repasse de um roteador consiste em um prefixo (por exemplo, 138.16.64/22) e uma porta de saída de roteador correspondente (por exemplo, porta 7). Quando um pacote chega ao roteador, o endereço IP de destino do pacote é comparado com os prefixos na tabela de repasse para achar aquele com a combinação de prefixo mais longa. O pacote é então repassado (dentro do roteador) para a porta do roteador associada a esse prefixo. Em seguida, vamos explicar como um registro de roteamento (prefixo e porta associada) é inserido em uma tabela de repasse. Este exercício simples juntará grande parte daquilo que aprendemos sobre roteamento e repasse. Para tornar as coisas mais interessantes, vamos supor que o prefixo seja um "prefixo externo", ou seja, ele não pertence ao AS do roteador, mas a algum outro AS.

Para que um prefixo entre na tabela de repasse do roteador, este precisa *primeiro conhecer* o prefixo (correspondente a uma sub-rede ou uma agregação de sub-redes). Como já vimos, o roteador passa a conhecer o prefixo por meio de um anúncio de rota BGP. Esse anúncio pode ser enviado a ele por uma sessão eBGP (de um roteador em outro AS) ou por uma sessão iBGP (de um roteador no mesmo AS).

Depois que o roteador conhece o prefixo, ele precisa determinar a porta de saída apropriada para onde os datagramas destinados a ele serão repassados, antes que possa entrar com esse prefixo na tabela de repasse. Se o roteador receber mais de um anúncio de rota para o prefixo, ele usa o processo de seleção de rota do BGP, conforme descrito antes nesta subseção, para achar a "melhor" rota. Suponha que essa melhor rota tenha sido selecionada. Como já dissemos, a rota selecionada inclui um atributo NEXT-HOP, que é o endereço IP do primeiro roteador fora do AS do roteador ao longo dessa melhor rota. Como vimos, o roteador usa então seu protocolo de roteamento intra-AS (em geral, OSPF) para determinar o caminho mais curto para o roteador do NEXT-HOP. O roteador por fim determina o número de porta para associar ao prefixo, identificando o primeiro enlace nesse caminho mais curto. O roteador pode, então (enfim!), entrar com o par prefixo-porta em sua tabela de repasse! A tabela de repasse calculada pelo processador de roteamento (ver Figura 4.6) é então colocada nas placas de linha da porta de entrada do roteador.

---

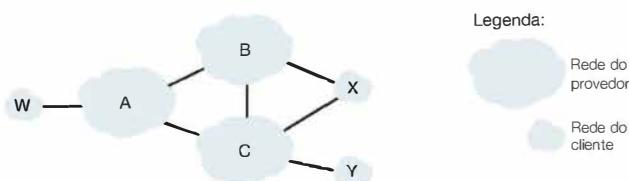
para o mesmo prefixo *x*. Essas duas rotas poderiam ter o mesmo AS-PATH para *x*, mas poderiam ter diferentes valores de NEXT-HOP correspondentes aos diferentes enlaces interconectados. Utilizando os valores de NEXT-HOP e o algoritmo de roteamento intra-AS, o roteador pode determinar o custo do caminho para cada enlace interconectado e, então, aplicar o roteamento batata quente (consulte Seção 4.5.3) para determinar a interface apropriada.

O BGP também inclui atributos que permitem que roteadores designem a métrica de sua preferência às rotas, e um atributo que indica como o prefixo foi inserido no BGP no AS de origem. Para obter uma discussão completa de atributos de rota, consulte Griffin [2012]; Stewart [1999]; Halabi [2000]; Feamster [2004]; RFC 4271.

Quando um roteador de borda recebe um anúncio de roteador, ele utiliza sua **política de importação** para decidir se aceita ou filtra a rota e se estabelece certos atributos, tal como a métrica de preferência do roteador. A política de importação pode filtrar uma rota porque o AS talvez não queira enviar tráfego por um dos ASs no AS-PATH. O roteador de borda também pode filtrar uma rota porque já conhece uma rota preferencial para o mesmo prefixo.

Seleção de rota do BGP

Como já descrito antes nesta seção, o BGP usa eBGP e iBGP para distribuir rotas a todos os roteadores dentro de ASs. Com essa distribuição, um roteador pode conhecer mais do que uma rota para qualquer prefixo determinado, quando deverá selecionar uma das possíveis. O dado usado por esse processo de seleção é o con-

junto de todas as rotas que o roteador descobriu e aceitou. Se houver duas ou mais para o mesmo prefixo, então o BGP invoca sequencialmente as seguintes regras de eliminação, até sobrar apenas uma:

- Rotas recebem, como um de seus atributos, um valor de preferência local. A preferência local de uma rota pode ter sido estabelecida pelo roteador ou ter sido descoberta por um outro roteador no mesmo AS. Essa é uma decisão política que fica a cargo do administrador de rede do AS. (Mais adiante discutiremos detalhes sobre questões de política do BGP.) São selecionadas as rotas que têm os valores de preferência local mais altos.
- No caso das outras rotas remanescentes (todas com o mesmo valor de preferência local), é selecionada a que tenha o AS-PATH mais curto. Se essa fosse a única regra de seleção, então o BGP estaria usando um algoritmo DV para determinação de caminho no qual a métrica da distância utiliza o número de saltos de AS em vez do número de saltos de roteadores.
- Dentre as rotas remanescentes (todas com o mesmo valor de preferência local e com o mesmo comprimento de AS-PATH) é selecionada a que tenha o roteador NEXT-HOP mais próximo. Nesse caso, mais próximo significa o que tenha o menor custo de caminho de menor custo, determinado pelo algoritmo intra-AS. Esse processo, como discutimos na Seção 4.5.3, é em geral denominado roteamento da batata quente.
- Se ainda restar mais de uma rota, o roteador usa identificadores BGP para selecionar a rota; veja Stewart, [1999].

As regras de eliminação são ainda mais complicadas do que descrevemos aqui. Para evitar pesadelos com o BGP, é melhor aprender suas regras de seleção em pequenas doses!

Política de roteamento

Vamos ilustrar alguns dos conceitos básicos da política de roteamento BGP com um exemplo simples. A Figura 4.42 mostra seis sistemas autônomos interconectados: A, B, C, W, X e Y. É importante notar que A, B, C, W, X e Y são ASs, e não roteadores. Vamos admitir que os sistemas autônomos W, X e Y são redes *stub* e que A, B e C são redes provedoras de *backbone*. Vamos supor também que A, B e C, todos conectados uns aos outros, fornecem informação completa sobre o BGP a suas redes cliente. Todo o tráfego que entrar em uma **rede *stub*** deve ser destinado a essa rede, e todo o tráfego que sair da rede *stub* deve ter sido originado naquela rede. W e Y são claramente redes *stub*. X é uma **rede *stub* com múltiplas interconexões**, visto que está ligado ao resto da rede por meio de dois provedores diferentes (um cenário que está se tornando cada vez mais comum na prática). Todavia, tal como W e Y, o próprio X deve ser a origem/destino de todo o tráfego que entra/sai de X. Porém, como esse comportamento da rede *stub* será executado e imposto? Como X será impedido de repassar tráfego entre B e C? Isso pode ser conseguido facilmente controlando o modo como as rotas BGP são anunciadas. Em particular, X funcionará como uma rede *stub* se anunciar (a seus vizinhos B e C) que não há nenhum caminho para quaisquer outros destinos a não ser ele mesmo. Isto é, mesmo que X conheça um caminho, digamos, XCY, que chegue até a rede Y, ele *não* anunciará esse caminho a B. Como B não fica sabendo que X tem um caminho para Y, B nunca repassaria tráfego destinado a Y (ou a C) por meio de X. Esse exemplo simples ilustra como uma política seletiva de anúncio de rota pode ser usada para implementar relacionamentos de roteamento cliente/provedor.

Em seguida, vamos focalizar uma rede provedora, digamos, o AS B. Suponha que B ficasse sabendo (por A) que A tem um caminho AW para W. Assim, B pode instalar a rota BAW em sua base de informações de roteamento. É claro que B também quer anunciar o caminho BAW a seu cliente, X, de modo que X saiba que pode rotear para W via B. Porém, B deveria anunciar o caminho BAW a C? Se o fizer, então C poderia rotear tráfego para W via CBAW. Se A, B e C forem todos provedores de *backbone*, então B poderia sentir-se no direito de achar que não deveria ter de suportar a carga (e o custo!) de transportar o tráfego em trânsito entre A e B. B poderia sentir-se no direito de achar que é de A e C o trabalho (e o custo!) de garantir que C possa rotear de/para clientes de A por meio de uma conexão direta entre A e C. Hoje não existe nenhum padrão oficial que determine como ISPs de *backbone*

FIGURA 4.42 UM CENÁRIO BGP SIMPLES

devem rotear entre si. Todavia, os ISPs comerciais adotam uma regra prática que diz que qualquer tráfego que esteja fluindo por uma rede de *backbone* de um ISP deve ter ou uma origem ou um destino (ou ambos) em uma rede que seja cliente daquele ISP; caso contrário, o tráfego estaria pegando uma carona gratuita na rede do ISP. Acordos individuais de parceria (*peering*) (para reger questões como as levantadas) costumam ser negociados entre pares de ISPs e, em geral, são confidenciais; Huston [1999a] provê uma discussão interessante sobre acordos de parceria. Se quiser uma descrição detalhada sobre como a política de roteamento reflete os relacionamentos comerciais entre

PRINCÍPIOS NA PRÁTICA

Por que há diferentes protocolos de roteamento inter-AS e intra-AS?

Agora que já examinamos os detalhes de protocolos de roteamento inter-AS e intra-AS específicos utilizados pela Internet, vamos concluir considerando a questão talvez mais fundamental que, antes de tudo, poderíamos levantar sobre esses protocolos (esperamos que você tenha estado preocupado com isso o tempo todo e que não tenha deixado de enxergar o quadro geral por causa dos detalhes!). Por que são usados diferentes protocolos de roteamento inter-AS e intra-AS?

A resposta a essa pergunta expõe o âmago da diferença entre os objetivos do roteamento dentro de um AS e entre ASs:

- *Política*. Entre ASs, as questões políticas dominam. Pode até ser importante que o tráfego que se origina em determinado AS não possa passar por outro AS específico. De maneira semelhante, determinado AS pode muito bem querer controlar o tráfego em trânsito que ele carrega entre outros ASs. Vimos que o BGP carrega atributos de caminho e oferece distribuição controlada de informação de roteamento, de modo que as decisões de roteamento baseadas em políticas possam ser tomadas. Dentro de um AS, tudo está nominalmente no mesmo controle administrativo. Assim, as questões de políticas de roteamento desempenham um papel bem menos importante na escolha de rotas no AS.
- *Escalabilidade*. A escalabilidade de um algoritmo de roteamento e de suas estruturas de dados para manipular o roteamento para/entre grandes números de redes é uma questão fundamental para o roteamento inter-AS. Dentro de um AS, a escalabilidade é uma preocupação menor. Isso porque, se um único domínio administrativo ficar muito grande, é sempre possível dividi-lo em dois ASs e realizar roteamento inter-AS entre esses dois novos ASs. (Lembre-se de que o OSPF permite que essa hierarquia seja construída dividindo um AS em áreas.)
- *Desempenho*. Dado que o roteamento inter-AS é bastante orientado pelas políticas, a qualidade (por exemplo, o desempenho) das rotas usadas é muitas vezes uma preocupação secundária (isto é, uma rota mais longa ou de custo mais alto que satisfaça a certos critérios políticos pode muito bem prevalecer sobre uma que é mais curta, mas que não satisfaz a esses critérios). Na verdade, vimos que entre ASs não há nem mesmo a ideia de custo associado às rotas (exceto a contagem de saltos do AS). Dentro de um AS individual, contudo, essas preocupações com as políticas têm menos importância, permitindo que o roteamento se concentre mais no nível de desempenho atingido em uma rota.

ISPs, veja Gao [2001]; Dmitiropoulos [2007]. Para ver uma abordagem recente sobre políticas de roteamento BGP, de um ponto de vista do ISP, consulte Caesar [2005b].

Como já observamos, o BGP é um padrão na prática para roteamento inter-AS na Internet pública. Para ver o conteúdo de várias tabelas de roteamento BGP (grandes!) extraídas de roteadores pertencentes a ISPs de nível 1, consulte <http://www.routeviews.org>. Tabelas de roteamento BGP em geral contêm dezenas de milhares de prefixos e atributos correspondentes. Estatísticas sobre tamanho e características de tabelas de roteamento BGP são apresentadas em Potaroo [2012].

Com isso concluímos nossa breve introdução ao BGP. Entender esse protocolo é importante porque ele desempenha um papel central na Internet. Aconselhamos você a consultar as referências Griffin [2012]; Stewart [1999]; Labovitz [1997]; Halabi [2000]; Huitema [1998]; Gao [2001]; Feamster [2004], Caesar [2005b]; Li [2007] para aprender mais sobre BGP.

4.7 ROTEAMENTO POR DIFUSÃO E PARA UM GRUPO

Até este ponto do capítulo, focalizamos protocolos de roteamento que suportam comunicação individual (isto é, ponto a ponto), em que um único nó de origem envia um pacote a um único nó de destino. Nesta seção, voltaremos a atenção a protocolos de roteamento por difusão (*broacast*) e para um grupo (*multicast*). No **roteamento por difusão** a camada de rede provê um serviço de entrega de pacote enviado de um nó de origem a todos os outros nós da rede; o **roteamento para um grupo** habilita um único nó de origem a enviar a cópia de um pacote a um subconjunto de nós das outras redes. Na Seção 4.7.1 consideraremos algoritmos de roteamento por difusão e sua incorporação em protocolos de roteamento. Examinaremos roteamento para um grupo na Seção 4.7.2.

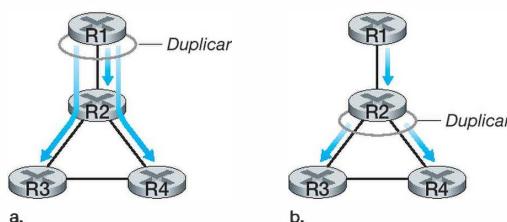
4.7.1 Algoritmos de roteamento por difusão (*broadcast*)

Talvez o modo mais direto de conseguir comunicação por difusão seja o nó remetente enviar uma cópia separada do pacote para cada destino, como mostra a Figura 4.43(a). Dados *N* nós de destino, o nó de origem simplesmente faz *N* cópias do pacote, endereça cada cópia a um destino diferente e então transmite *N* cópias aos *N* destinos usando roteamento individual. Essa abordagem **individual de N caminhos** da transmissão por difusão é simples — não é preciso nenhum novo protocolo de roteamento de camada de rede, nem duplicação de pacotes, nem funcionalidade de repasse. Porém, tem várias desvantagens. A primeira é sua ineficiência. Se o nó de origem estiver conectado ao resto da rede por um único enlace, então *N* cópias separadas do (mesmo) pacote transitarão por esse único enlace. Evidentemente seria mais eficiente enviar apenas uma única cópia de um pacote até esse primeiro salto e então fazer o nó na outra extremidade do primeiro salto produzir e transmitir quaisquer cópias adicionais necessárias. Isto é, seria mais eficiente que os próprios nós da rede (em vez de apenas o nó de origem) criassem cópias duplicadas de um pacote. Por exemplo, na Figura 4.43(b), somente uma única cópia de um pacote transita pelo enlace R1-R2. Aquele pacote, então, é duplicado em R2, e uma única cópia será enviada pelos enlaces R2-R3 e R2-R4.

As desvantagens adicionais da abordagem individual de *N* caminhos talvez sejam mais sutis, mas nem por isso menos importantes. Uma premissa implícita desse tipo de roteamento é que o remetente conheça os destinatários de difusão e seus endereços. No entanto, como essa informação é obtida? Muito provavelmente, seriam exigidos mecanismos de protocolo adicionais (tais como associação ao grupo de difusão ou protocolo de registro de destino), o que acrescentaria sobrecarga e, o que é importante, complexidade adicional a um protocolo que, de início, parecia bastante simples. Uma desvantagem final da abordagem individual de *N* caminhos está relacionada aos propósitos para os quais a difusão seria utilizada. Na Seção 4.5 aprendemos que protocolos de roteamento de estado de enlace usam difusão para propagar informações de estado de enlace que são usadas para calcular rotas individuais. É claro que, em situações em que o grupo de difusão é usado para criar e atualizar rotas individuais, seria imprudente (no mínimo!) confiar em infraestrutura de roteamento individual para implantar grupos de difusão.

Dadas as diversas desvantagens da difusão de N caminhos, logicamente são de interesse abordagens nas quais os próprios nós da rede desempenham um papel ativo na duplicação de pacotes, no repasse de pacotes e no cálculo de rotas de difusão. A seguir, examinaremos diversas dessas abordagens e, mais uma vez, adotaremos a notação de grafo apresentada na Seção 4.5. Modelaremos novamente a rede como um grafo $G = (N, E)$, onde N é um conjunto de nós e uma coleção E de arestas, e cada aresta é um par de nós de N. Não seremos muito exigentes quanto à notação e adotaremos N para indicar ambos os conjuntos de nós, e a cardinalidade ($|N|$) ou o tamanho daquele conjunto, quando não houver possibilidade de confusão.

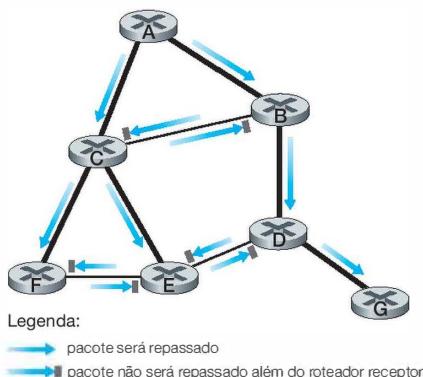
Inundação não controlada

A técnica mais óbvia para conseguir difusão é uma abordagem de **inundação** na qual o nó de origem envia uma cópia do pacote a todos os seus vizinhos. Quando um nó recebe um pacote de difusão, ele o duplica e repassa a todos os seus vizinhos (exceto ao vizinho do qual recebeu o pacote). É claro que, se o grafo for conectado, esse esquema enfim entregará uma cópia do pacote de difusão a todos os nós no grafo. Embora esse esquema seja simples e elegante, tem uma falha fatal (antes de continuar, tente imaginar qual): se o grafo tiver ciclos, então uma ou mais cópias de cada pacote de difusão permanecerão em ciclo indefinidamente. Por exemplo, na Figura 4.43, R2 inundará R3, R3 inundará R4, R4 inundará R2 e R2 inundará (novamente!) R3 e assim por diante. Esse cenário simples resulta no ciclo sem fim de dois pacotes de difusão, um em sentido horário e um em sentido anti-horário. Mas pode haver uma falha fatal ainda mais catastrófica: quando um nó estiver conectado a mais de dois outros nós, criará e repassará várias cópias do pacote de difusão, cada uma das quais criará várias cópias de si mesmas (em outros nós com mais de dois vizinhos) e assim por diante. Essa **tempestade de difusão**, gerada pela infindável multiplicação depacotes de difusão, acabaria resultando na criação de uma quantidade tão grande de pacotes de difusão que a rede ficaria inutilizada. (Consulte nos exercícios de fixação, ao final do capítulo, um problema que analisa a velocidade com que essa tempestade de difusão cresce.)

Inundação controlada

A chave para evitar uma tempestade de difusão é um nó escolher de modo sensato quando repassa um pacote (por exemplo, se já tiver recebido e repassado uma cópia anterior do pacote) e quando não repassa um pacote. Na prática, há vários modos de fazer isso.

Na **inundação controlada por número de sequência**, um nó de origem coloca seu endereço (ou outro identificador exclusivo), bem como um **número de sequência de difusão** em um pacote de difusão e então envia o pacote a todos os seus vizinhos. Cada nó mantém uma lista de endereços de origem e números de sequência para cada pacote de difusão que já recebeu, duplicou e repassou. Quando um nó recebe um pacote de difusão, primeiro verifica se o pacote está nessa lista. Se estiver, é descartado; se não estiver, é duplicado e repassado para todos os vizinhos do nó (exceto para o nó de quem o pacote acabou de ser recebido). O protocolo Gnutella, discutido no

FIGURA 4.43 DUPLICAÇÃO NA ORIGEM *VERSUS* DUPLICAÇÃO DENTRO DA REDE

Capítulo 2, usa inundação controlada com número de sequência para fazer a transmissão por difusão de consultas em sua rede de sobreposição. (Em Gnutella, a duplicação e repasse de mensagens é realizada na camada de aplicação, e não na camada de rede.)

Uma segunda abordagem da inundação controlada é conhecida como **repasse pelo caminho inverso** (*reverse path forwarding* — RPF) [Dalal, 1978], às vezes também denominado difusão pelo caminho inverso (*reverse path broadcast* — RPB). A ideia por trás do repasse pelo caminho inverso é simples, mas elegante. Quando um roteador recebe um pacote de difusão com determinado endereço de origem, ele transmite o pacote para todos os seus enlaces de saída (exceto para aquele do qual o pacote foi recebido) somente se o pacote chegou pelo enlace que está em seu próprio caminho individual mais curto de volta ao remetente. Caso contrário, o roteador apenas descarta o pacote que está entrando sem repassá-lo para nenhum de seus enlaces de saída. O pacote pode ser descartado porque o roteador sabe que receberá, ou já recebeu, uma cópia no enlace que está em seu próprio caminho mais curto de volta ao remetente. (É provável que você esteja querendo se convencer de que isso, de fato, acontecerá, e que não ocorrerão nem *looping* nem tempestades de difusão.) Note que o RPF não usa roteamento de difusão individual para entregar um pacote a um destino, nem exige que um roteador conheça o caminho mais curto completo entre ele mesmo e a origem. O RPF precisa conhecer apenas o próximo vizinho em seu caminho individual mais curto até o remetente; usa a identidade desse vizinho apenas para determinar se repassa ou não repassa um pacote de difusão recebido.

A Figura 4.44 ilustra o RPF. Suponha que os enlaces com as linhas mais grossas representem os caminhos de menor custo desde os destinatários até a origem (*A*). O nó *A* inicialmente faz a transmissão por difusão de um pacote da origem *A* até os nós *C* e *B*. O nó *B* repassará o pacote da origem *A* que recebeu de *A* (uma vez que *A* está em seu caminho de menor custo até *A*) para *C* e *D*. *B* ignorará (descartará sem repassar) quaisquer pacotes da origem *A* que receba de quaisquer outros nós (por exemplo, dos roteadores *C* ou *D*). Vamos considerar agora o nó *C*, que receberá um pacote da origem *A* direto de *A* e também de *B*. Como *B* não está no caminho mais curto de *C* de retorno a *A*, *C* ignorará quaisquer pacotes da origem *A* que recebam de *B*. Por outro lado, quando *C* receber um pacote da origem *A* direto de *A*, ele o repassará aos nós *B*, *E* e *F*.

Difusão por *spanning tree*

Conquanto a inundação controlada por números de sequência e o RPF impeçam tempestades de difusão, eles não evitam completamente a transmissão de pacotes por difusão redundantes. Por exemplo, na Figura 4.44, os nós *B*, *C*, *D* e *F* recebem ou um ou dois pacotes redundantes. Idealmente, cada nó deveria receber apenas uma cópia do pacote de difusão. Examinando a árvore formada pelos nós conectados por linhas grossas na Figura 4.45(a), podemos ver que, se os pacotes de difusão fossem repassados por enlaces dessa árvore, cada

FIGURA 4.44 REPASSE PELO CAMINHO INVERSO

nó da rede receberia exatamente uma cópia do pacote de difusão — exatamente a solução que procurávamos! Esse é um exemplo de ***spanning tree*** — uma árvore que contém todo e qualquer nó em um grafo. Mais formalmente, uma *spanning tree* de um grafo $G = (N, E)$ é um grafo $G' = (N, E')$ tal que E' é um subconjunto de E, G' é conectado, G' não contém nenhum ciclo e G' contém todos os nós originais em G. Se cada enlace tiver um custo associado e o custo de uma árvore for a soma dos custos dos enlaces, então uma árvore cujo custo seja o mínimo entre todas as *spanning trees* do gráfico é denominada uma ***spanning tree* mínima** (o que não é nenhuma surpresa).

Assim, outra abordagem para o fornecimento de difusão é os nós da rede construírem uma *spanning tree*, em primeiro lugar. Quando um nó de origem quiser enviar um pacote por difusão, enviará o pacote por todos os enlaces incidentes que pertençam à *spanning tree*. Um nó que receba um pacote por difusão então o repassa a todos os seus vizinhos na *spanning tree* (exceto para o vizinho do qual recebeu o pacote). A *spanning tree* não apenas elimina pacotes por difusão redundantes mas, uma vez instalada, pode ser usada por qualquer nó para iniciar uma difusão, como mostram as figuras 4.45(a) e 4.45(b). Note que um nó não precisa estar ciente da árvore inteira; ele precisa apenas saber qual de seus vizinhos em G são vizinhos que pertencem à *spanning tree*.

A principal complexidade associada com a abordagem de *spanning tree* é sua criação e manutenção. Foram desenvolvidos numerosos algoritmos distribuídos de *spanning tree* [Gallager, 1983; Gartner, 2003]. Aqui, consideramos apenas um algoritmo simples. Na **abordagem de nó central** da construção de uma *spanning tree*, é definido um nó central (também conhecido como **ponto de encontro** ou **núcleo**). Os nós então transmitem mensagens de adesão individualmente à árvore, endereçadas ao nó central. Uma mensagem de adesão à árvore é repassada usando roteamento individual em direção ao centro até que ela chegue a um roteador que já pertence à *spanning tree* ou até que chegue ao centro. Em qualquer um dos casos, o caminho que a mensagem de adesão à árvore seguiu define o ramo da *spanning tree* entre o nó da borda que enviou a mensagem de adesão à árvore e o centro. Esse novo caminho pode ser visto como um enxerto à *spanning tree* existente.

A Figura 4.46 ilustra a construção de uma *spanning tree* baseada em um centro. Suponha que o nó E seja selecionado como centro da árvore. Imagine que o nó F primeiro se junte à árvore e repasse a E uma mensagem de adesão à árvore. O único enlace EF se torna a *spanning tree* inicial. O nó B então se junta à *spanning tree* enviando a E sua mensagem de adesão à árvore. Suponha que a rota do caminho individual de E para B seja por D. Nesse caso, a mensagem de adesão à árvore resulta no enxerto do caminho BDE à *spanning tree*. Em seguida, o nó A se junta ao grupo crescente repassando sua mensagem de adesão à árvore em direção a E. Se o caminho individual de A a E passar por B, então, uma vez que B já se juntou à *spanning tree*, a chegada da mensagem de adesão à árvore de A em B resultará no enxerto imediato do enlace AB à *spanning tree*. Em seguida, o nó C se junta à *spanning tree* repassando sua mensagem de adesão à árvore diretamente a E. Por fim, como o roteamento individual de G para E tem de passar pelo nó D, quando G enviar sua mensagem de adesão à árvore a E, o enlace GD será enxertado à *spanning tree* no nó D.

FIGURA 4.45 DIFUSÃO POR *SPANNING TREE*

a. Difusão iniciada em *A*

b. Difusão iniciada em *D*

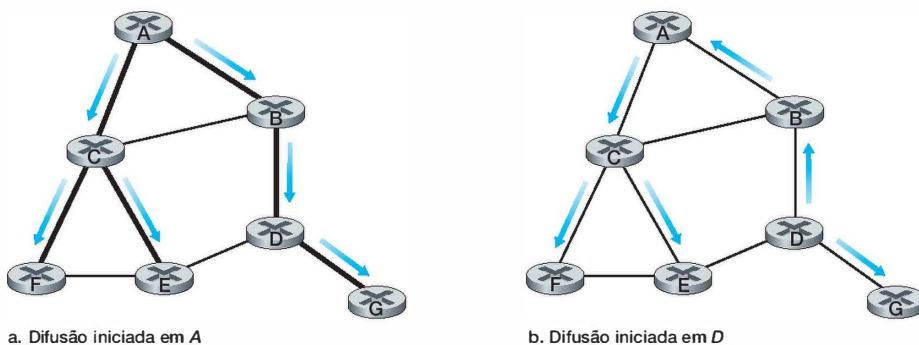
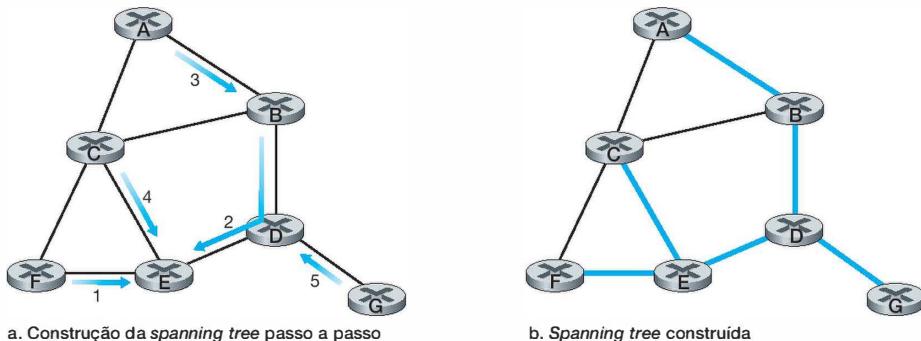



FIGURA 4.46 CONSTRUÇÃO DE UMA *SPANNING TREE* COM CENTRO

a. Construção da *spanning tree* passo a passo

b. *Spanning tree* construída

Algoritmos de difusão na prática

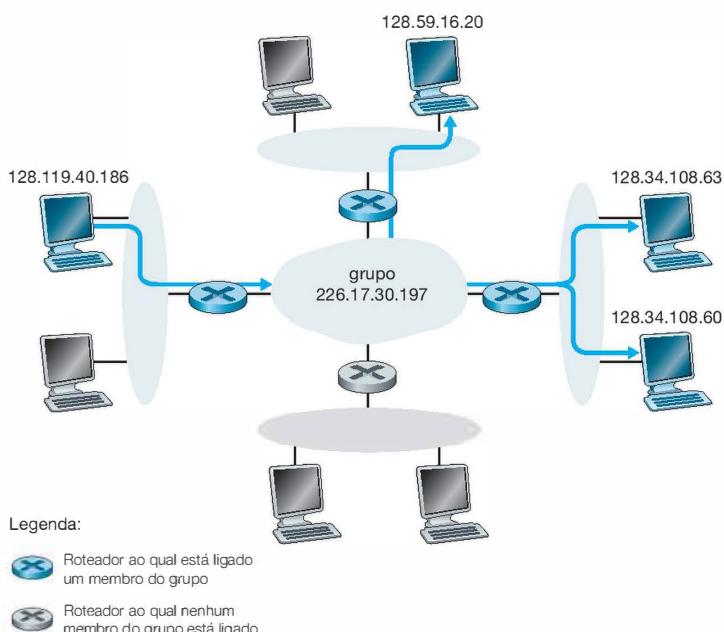
Protocolos de difusão são usados na prática nas camadas de aplicação e de rede. Gnutella [2009] usa a difusão no nível de aplicação para fazer transmissão por difusão de consultas de conteúdo entre pares Gnutella. Nesse caso, um enlace entre dois processos pares distribuídos no nível de aplicação na rede Gnutella é, na verdade, uma conexão TCP. O Gnutella usa uma forma de inundação controlada com número de sequência na qual um identificador de 16 bits e um descritor de carga útil de 16 bits (que identifica o tipo de mensagem Gnutella) são usados para detectar se uma consulta por difusão recebida já foi recebida, duplicada e repassada antes. O Gnutella usa um campo de tempo de vida (TTL) para limitar o número de saltos pelos quais uma consulta inundada será repassada. Quando um processo Gnutella recebe e duplica uma consulta, ele decrementa o campo de TTL antes de repassá-la. Assim, uma consulta Gnutella inundada alcançará apenas pares que estão dentro de determinado número (o valor inicial do TTL) de saltos em nível de aplicação desde o iniciador da consulta. Por isso, o mecanismo de consultas por inundação do Gnutella às vezes é denominado *inundação de escopo limitado*.

Uma forma de inundação controlada com número de sequência também é usada para fazer a transmissão por difusão de anúncios de estado de enlace (*link-state advertisements* — LSAs) no algoritmo de roteamento OSPF [RFC 2328; Perlman, 1999] e no algoritmo de roteamento Sistema Intermediário a Sistema Intermediário (Intermediate-System-to-Intermediate-System — IS-IS) [RFC 1142; Perlman, 1999]. O OSPF usa um número de sequência de 32 bits, bem como um campo de idade de 16 bits para identificar anúncios de estado de enlace (LSAs). Lembre-se de que um nó OSPF faz *broadcast* periódico de LSAs para os enlaces ligados a ele quando o custo de enlace até um vizinho muda ou quando um enlace ativa ou desativa. Números de sequência LSA são usados para detectar LSAs duplicados, mas também cumprem uma segunda função importante no OSPF. Com a inundação, é possível que um LSA gerado pela origem no tempo t chegue *após* um LSA mais novo que foi gerado pela mesma origem no tempo $t + \delta$. Os números de sequência usados pelo nó da origem permitem que um LSA mais antigo seja distinguido de um mais novo. O campo de idade cumpre uma finalidade semelhante à de um valor TTL. O valor inicial do campo de idade é estabelecido em zero e é incrementado a cada salto à medida que este é retransmitido, e também é incrementado enquanto fica na memória de um roteador esperando para ser inundado. Embora nossa descrição do algoritmo de inundação LSA tenha sido apenas breve, observamos que, na verdade, projetar protocolos de difusão LSA pode ser um negócio muito delicado. RFC 789 e Perlman [1999] descrevem um incidente no qual LSAs transmitidos incorretamente por dois roteadores defeituosos fizeram uma versão antiga do algoritmo de inundação LSA derrubar a ARPAnet inteira!

4.7.2 Serviço para um grupo (*multicast*)

Vimos, na seção anterior, que, com o serviço de difusão, pacotes são entregues a cada um e a todos os nós em uma rede. Nesta seção, voltamos nossa atenção para o serviço **para um grupo**, no qual um pa-

cote é entregue a apenas um *subgrupo* de nós de rede. Uma série de aplicações emergentes de rede requer a entrega de pacotes de um ou mais remetentes a um grupo de destinatários. Entre essas aplicações estão a transferência de dados em grandes volumes (por exemplo, a transferência de uma atualização de software do desenvolvedor do software para os usuários que necessitam da atualização), mídia de fluxo contínuo (por exemplo, transferência de áudio, vídeo e texto de uma palestra ao vivo para um conjunto de participantes distribuídos), as aplicações de dados compartilhados (por exemplo, quadro branco ou videoconferência, compartilhados por muitos participantes distribuídos), a alimentação de dados (por exemplo, cotação de ações), a atualização de *cache* da Web e os jogos interativos (por exemplo, ambientes virtuais interativos distribuídos ou jogos multiusuários).

Na comunicação para um grupo, enfrentamos imediatamente dois problemas — como identificar os destinatários de um pacote desse tipo e como endereçar um pacote enviado a um desses destinatários. No caso da comunicação individual, o endereço IP do destinatário (receptor) é levado em cada datagrama IP individual e identifica o único destinatário; no caso do serviço para um grupo, temos vários destinatários. Tem sentido que cada pacote desse tipo carregue os endereços IP de todos os vários destinatários? Embora essa abordagem possa ser funcional com um número pequeno de destinatários, não pode ser expandida com facilidade para o caso de centenas de milhares de destinatários; a quantidade de informações de endereçamento no datagrama ultrapassaria a quantidade de dados que é de fato carregada no campo de carga útil do pacote. A identificação explícita dos destinatários pelo remetente também exige que o remetente conheça as identidades e os endereços de todos os destinatários. Veremos em breve que há casos em que essa exigência pode ser indesejável.

Por essas razões, na arquitetura da Internet (e em outras arquiteturas de rede como ATM [Black, 1995]), um pacote para um grupo é endereçado usando **endereço indireto**, isto é, um único identificador é utilizado para o grupo de destinatários e uma cópia do pacote que é endereçada ao grupo usando esse único identificador é entregue a todos os destinatários associados ao grupo. Na Internet, o identificador único que representa um grupo de destinatários é um endereço IP classe D para um grupo. O grupo de destinatários associados a um endereço classe D é denominado **grupo *multicast***. A abstração do grupo é ilustrada na Figura 4.47. Na figura, quatro hospedeiros (no tom mais claro) estão associados ao endereço de grupo 226.17.30.197 e receberão todos os datagramas endereçados a esse endereço de grupo. A dificuldade que ainda temos de enfrentar é o fato de que cada hospedeiro tem um endereço individual exclusivo que é completamente independente do endereço do grupo do qual ele está participando.

Embora seja simples, a abstração do grupo provoca uma série de perguntas. Como um grupo começa e como termina? Como é escolhido o endereço? Como novos hospedeiros são incorporados (seja como remetentes, seja como destinatários)? Qualquer um pode se juntar ao grupo (e enviar para esse grupo e receber dele) ou a participação no grupo é limitada e, se for limitada, quem a limita? Os membros do grupo ficam conhecendo as identidades dos outros membros como parte do protocolo de camada de rede? Como os nós da rede interagem para entregar um datagrama de grupo a todos os membros desse grupo? Para a Internet, as respostas a todas essas perguntas envolvem o Protocolo de Gerenciamento de Grupo da Internet (*Internet Group Management Protocol* — IGMP [RFC 3376]). Assim, vamos considerar rapidamente o protocolo IGMP. Mais adiante, voltaremos a essas perguntas de caráter mais geral.

Internet Group Management Protocol

O protocolo IGMP, versão 3 [RFC 3376] opera entre um hospedeiro e o roteador diretamente conectado a ele (em termos informais, pense no roteador diretamente conectado ao hospedeiro como o roteador de primeiro salto que um hospedeiro veria no caminho até qualquer hospedeiro externo à sua própria rede local, ou como o roteador de último salto em qualquer caminho até esse hospedeiro), conforme mostra a Figura 4.48. A figura mostra os três roteadores do primeiro salto, cada um conectado ao hospedeiro ao qual está ligado por uma interface local de saída. Essa interface local está ligada a uma LAN nesse exemplo e, embora cada LAN tenha vários

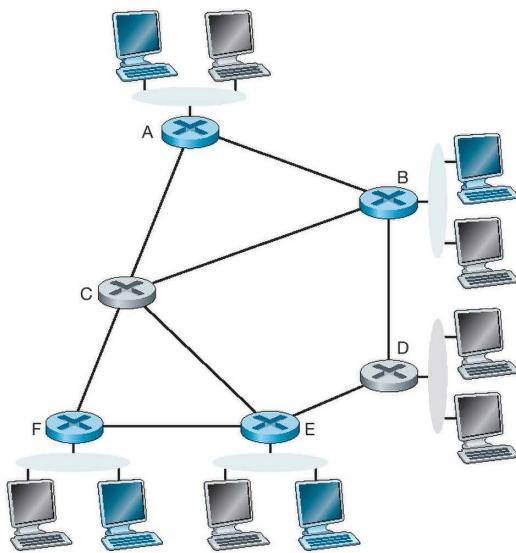
FIGURA 4.47 O SERVIÇO PARA UM GRUPO: UM DATAGRAMA ENDEREÇADO AO GRUPO É ENTREGUE A TODOS OS MEMBROS DO GRUPO

hospedeiros ligados a ela, no máximo apenas alguns deles comumente pertencerão a um dado grupo a qualquer determinado instante.

O IGMP oferece os meios para um hospedeiro informar ao roteador conectado a ele que uma aplicação que está funcionando no hospedeiro quer se juntar a um grupo específico. Dado que o escopo de interação do IGMP é limitado a um hospedeiro e seu roteador conectado, outro protocolo é decerto necessário para coordenar os roteadores de grupo (incluindo os conectados) por toda a Internet, de modo que os datagramas de grupo sejam roteados a seus destinos finais. Esta última funcionalidade é realizada por algoritmos de roteamento de grupo da camada de rede, como os que veremos em breve. Assim, o serviço de grupo na camada de rede da Internet consiste em dois componentes complementares: IGMP e protocolos de roteamento de grupo.

O IGMP tem apenas três tipos de mensagens. Como as mensagens ICMP, as mensagens IGMP são transportadas (encapsuladas) dentro de um datagrama IP, com um número de protocolo igual a 2. A mensagem `membership_query` é enviada por um roteador a todos os hospedeiros em uma interface conectada (por exemplo, para todos os hospedeiros em uma rede local) para determinar o conjunto de todos os grupos aos quais se ligaram os hospedeiros naquela interface. Os hospedeiros respondem a uma mensagem `membership_query` com uma mensagem IGMP `membership_report`. As mensagens `membership_report` também podem ser geradas por um hospedeiro quando uma aplicação se junta pela primeira vez ao grupo sem esperar por uma mensagem `membership_query` vinda do roteador. O último tipo de mensagem IGMP é a mensagem `leave_group`. O interessante é que essa mensagem é opcional. Mas, se é opcional, como um roteador detecta quando um hospedeiro sai do grupo? A resposta é que o roteador *deduz* que o hospedeiro não está mais ligado a determinado grupo quando nenhum hospedeiro responde a uma mensagem `membership_query` com esse endereço de grupo. Isso é um exemplo do que às vezes é denominado **estado flexível** em um protocolo da Internet. Em um protocolo de estado flexível, o estado (no caso do

FIGURA 4.48 OS DOIS COMPONENTES DE GRUPO DA CAMADA DE REDE: IGMP E PROTOCOLOS DE ROTEAMENTO PARA UM GRUPO

IGMP, o fato de haver hospedeiros ligados a um dado grupo) será encerrado por um evento de esgotamento de temporização (nesse caso, por uma mensagem `membership_query` periódica emitida pelo roteador) se não for explicitamente renovado (nesse caso, por uma mensagem `membership_report` vinda de um hospedeiro conectado).

O termo estado flexível (*soft state*) foi criado por Clark [1988], que descreveu a noção de mensagens periódicas de atualização de estado enviadas por um sistema final, e sugeriu que, com tais mensagens, o estado poderia ser perdido em uma falha qualquer e depois restaurado automaticamente por mensagens de renovação subsequentes — tudo de forma transparente ao sistema final e sem chamar qualquer procedimento explícito de recuperação de falhas:

> *(...) a informação de estado não seria crítica na manutenção do tipo desejado de serviço associado ao fluxo. Em vez disso, esse tipo de serviço seria imposto pelas extremidades, que periodicamente enviariam mensagens para garantir que o tipo de serviço apropriado estava sendo associado ao fluxo. Desse modo, a informação de estado associada ao fluxo poderia ser perdida em uma falha qualquer sem a interrupção permanente dos recursos sendo utilizados. Chamo esse conceito de "estado flexível", e ele pode muito bem nos permitir alcançar nossos objetivos principais de capacidade de sobrevivência e flexibilidade...*

Argumenta-se que os protocolos de estado flexível resultam em um controle mais simples do que os de estado rígido, que não apenas exigem que o estado seja acrescentado e removido explicitamente, mas também mecanismos para se recuperar da situação onde a entidade responsável por remover o estado tenha terminado prematuramente ou falhado de alguma maneira. Discussões interessantes sobre o estado flexível podem ser vistas em Raman [1999]; Ji [2003]; Lui [2004].

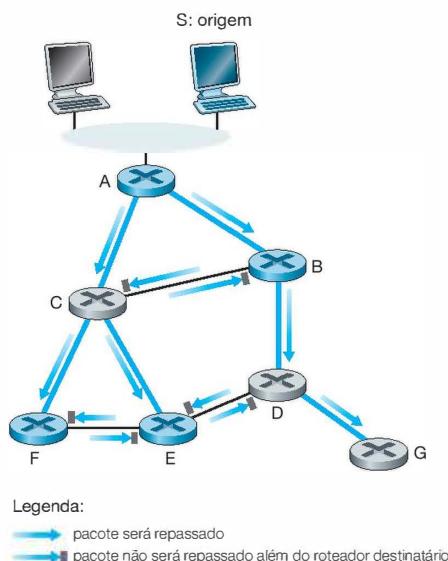
Algoritmos de roteamento para grupos

O **problema de roteamento para grupos** (*multicast*) está ilustrado na Figura 4.49. Os hospedeiros agregados a esse grupo são os de tom mais claro; os roteadores imediatamente conectados a eles também estão nesse tom. Como mostra a figura, apenas um subconjunto desses roteadores (os que têm hospedeiros conectados que se juntaram ao grupo) realmente precisa receber o tráfego para o grupo. Ou seja, apenas os roteadores *A*, *B*, *E* e *F* precisam receber esse tráfego. Uma vez que nenhum dos hospedeiros conectados ao roteador *D* está ligado ao grupo e o roteador *C* não tem hospedeiros conectados, nem *C* nem *D* precisam receber o tráfego do grupo. A meta do roteamento para um grupo é encontrar uma árvore de enlaces que conecte todos os roteadores que têm hospedeiros conectados pertencentes ao grupo. Então, pacotes para um grupo serão roteados ao longo dessa

FIGURA 4.49 HOSPEDEIROS DO GRUPO, SEUS ROTEADORES CONECTADOS E OUTROS ROTEADORES

árvore desde o remetente até todos os hospedeiros pertencentes à árvore de grupo. Claro, a árvore pode conter roteadores que não têm hospedeiros conectados pertencentes ao grupo (por exemplo, na Figura 4.48, é impossível conectar os roteadores *A*, *B*, *E* e *F* em uma árvore sem envolver o roteador *C* ou *D*).

Na prática, duas abordagens foram adotadas para determinar a árvore de roteamento para um grupo e ambas já foram estudadas no contexto do roteamento por difusão, portanto vamos apenas mencioná-las. A diferença entre as duas abordagens refere-se ao uso de uma única árvore compartilhada pelo grupo para distribuir o tráfego para todos os remetentes do grupo ou à construção de uma árvore de roteamento específica para cada remetente individual:

- *Roteamento multicast usando uma árvore compartilhada pelo grupo*. Como no caso da difusão por *spanning tree*, o roteamento para um grupo por uma árvore compartilhada por esse grupo é baseado na construção de uma árvore que inclui todos os roteadores de borda cujos hospedeiros a ele conectados pertencem ao grupo. Na prática, é usada uma abordagem centralizada para construir a árvore de roteamento para o grupo — roteadores de aresta cujos hospedeiros a eles ligados pertencem ao grupo enviam (individualmente) mensagens de adesão endereçadas ao nó central. Como no caso da difusão, uma mensagem de adesão é transmitida usando roteamento individual em direção ao centro até que a mensagem chegue a um roteador que já pertence à árvore de grupo ou chegue até o centro. Todos os roteadores do caminho percorrido pela mensagem de adesão então transmitirão os pacotes recebidos pelo grupo ao roteador de aresta que iniciou a adesão. Uma questão crítica para o roteamento para um grupo por meio de árvore baseado em um centro é o processo utilizado para selecionar o centro. Algoritmos de seleção de centro são discutidos em Wall [1980]; Thaler [1997]; Estrin [1997].
- *Roteamento para um grupo usando uma árvore baseada na origem*. Enquanto o roteamento para um grupo por meio de árvore compartilhada pelo grupo constrói uma única árvore compartilhada para rotear pacotes de *todos* os remetentes, a segunda abordagem constrói uma árvore de roteamento para um grupo para *cada origem* no grupo. Na prática, é usado um algoritmo RPF (com nó de origem *x*) para construir uma árvore de transmissão para um grupo a fim de enviar datagramas que venham da origem *x*. Para ser usado em um grupo, o algoritmo RPF que estudamos precisa sofrer alguma adaptação. Para ver por que, considere o roteador *D* na Figura 4.50. Com difusão RPF, o algoritmo repassaria pacotes ao roteador *G*,

FIGURA 4.50 REPASSE PELO CAMINHO INVERSO, NO CASO DO SERVIÇO PARA UM GRUPO

mesmo que este não tivesse nenhum hospedeiro conectado a ele pertencente ao grupo. Embora isso não seja tão ruim para este caso, em que *D* tem apenas um roteador adiante dele, *G*, imagine o que aconteceria se houvesse milhares de roteadores adiante de *D*! Cada um dos milhares de roteadores receberia pacotes indesejados para um grupo. (Esse cenário não está tão longe da realidade como parece. A Mbone inicial [Casner, 1992; Macedonia, 1994], a primeira rede de grupo global, sofria exatamente desse problema no início.) A solução para o problema do recebimento de pacotes de grupo indesejados sob RPF é conhecida como **poda**. Um roteador para um grupo que receba pacotes de grupo e não tenha nenhum hospedeiro conectado a ele pertencente àquele grupo enviará uma mensagem de poda ao roteador anterior a ele. Se um roteador receber mensagens de poda de cada um dos roteadores que estão adiante dele, então pode repassar uma mensagem de poda aos que estão antes dele.

Roteamento para grupos na Internet

O primeiro protocolo de roteamento para grupos usado na Internet foi o **Protocolo de Roteamento para um Grupo por Vetor de Distâncias** (**Distance-Vector Multicast Routing Protocol — DVMRP**) [RFC 1075]. O DVMRP executa árvores específicas de origem com repasse de caminho inverso e poda. Esse protocolo usa um algoritmo RPF com poda, como acabamos de ver. Talvez o protocolo de roteamento para um grupo mais utilizado na Internet seja o protocolo de roteamento para um **grupo independente de protocolo** (**PIM — Protocol-Independent Multicast**), o qual reconhece explicitamente dois cenários de distribuição para um grupo. No modo denso [RFC 3973], os membros do grupo *multicast* são densamente localizados; ou seja, muitos ou a maioria dos roteadores na área precisam estar envolvidos nos datagramas de roteamento para um grupo. O PIM de modo denso é uma técnica de repasse de caminho inverso do tipo “inundar e podar”, semelhante em espírito ao DVMRP.

No modo esparso [RFC 4601], o número de roteadores com membros do grupo é pequeno em relação ao número total de roteadores; os membros do grupo são amplamente dispersos. O PIM de modo esparso usa pontos de encontro para definir a árvore de distribuição *multicast*. No **grupo de origem específica** (**SSM —**

source-specific multicast) [RFC 3569, RFC 4607], só um remetente pode enviar tráfego para a árvore de grupo, simplificando de modo considerável a construção e a manutenção da árvore.

Quando o PIM e o DVMRP são usados dentro de um domínio, o operador de rede pode configurar roteadores IP de grupo dentro do domínio, da mesma forma que protocolos de roteamento individual intradomínio como RIP, IS-IS e OSPF podem ser configurados. Mas o que acontece quando são necessárias rotas de grupo entre domínios diferentes? Existe algum equivalente ao protocolo BGP interdomínio para grupos? A resposta é (literalmente) sim. [RFC 4271] define as extensões de protocolos múltiplos para o BGP que lhe permite carregar informações sobre roteamento para outros protocolos, incluindo informações de grupo. O protocolo MSDP (Multicast Source Discovery Protocol) [RFC 3618, RFC 4611] pode ser usado para conectar os pontos de encontro em domínios de diferentes PIMs de modo esparso. Uma visão geral interessante sobre o atual estado do roteamento para um grupo na Internet é o [RFC 5110].

Vamos finalizar nossa discussão sobre IP para um grupo observando que ele ainda tem de decolar de forma significativa. Para ver discussões interessantes sobre o modelo de serviço de grupo da Internet e questões sobre implementação, consulte Diot [2000]; Sharma [2003]. Não obstante, apesar da falta de emprego generalizado, o uso de grupos em nível de rede está longe de "morrer". O tráfego de grupo foi conduzido por muitos anos na Internet 2, assim como as redes com as quais ela se interconecta [Internet2 Multicast, 2012]. No Reino Unido, a BBC está envolvida em testes de distribuição de conteúdo pelo uso do IP de grupo [BBC Multicast, 2012]. Ao mesmo tempo, o uso de grupos em nível de aplicação, como vimos em PPLive, no Capítulo 2, e outros sistemas *peer-to-peer*, como o End System Multicast [Chu, 2002], oferecem distribuição de conteúdo para um grupo entre pares que utilizam protocolos de grupo da camada de aplicação (em vez da camada de rede). Os serviços de grupo do futuro serão primeiro executados na camada de rede (no núcleo da rede) ou na de aplicação (na borda da rede)? Enquanto a atual mania de distribuição de conteúdo *peer-to-peer* influencia a favor da camada de aplicação de grupo pelo menos em um futuro próximo, a evolução continua a caminhar no IP de grupo, e às vezes, no fim das contas, devagar se vai ao longe.

4.8 RESUMO

Neste capítulo, iniciamos nossa viagem rumo ao núcleo da rede. Aprendemos que a camada de rede envolve todos os roteadores da rede. Por causa disso, os protocolos de camada de rede estão entre os mais desafiadores da pilha de protocolos.

Aprendemos que um roteador pode precisar processar milhões de fluxos de pacotes entre diferentes pares origem-destino ao mesmo tempo. Para que um roteador consiga processar toda essa quantidade de fluxos, os projetistas de redes aprenderam, com o passar dos anos, que as tarefas do roteador devem ser as mais simples possíveis. Muitas medidas podem ser tomadas para facilitar a tarefa do roteador, incluindo a utilização de uma camada de rede de datagramas, em vez de uma camada de rede de circuitos virtuais, a utilização de um cabeçalho aprimorado de tamanho fixo (como no IPv6), a eliminação da fragmentação (também feita no IPv6) e o fornecimento de um único serviço de melhor esforço. Talvez o truque mais importante aqui seja *não* monitorar fluxos individuais, mas, em vez disso, tomar decisões de roteamento com base exclusivamente em endereços de destino hierarquicamente estruturados nos datagramas. É interessante notar que o serviço dos correios vem usando essa mesma tática há muitos anos.

Neste capítulo, examinamos também os princípios básicos dos algoritmos de roteamento. Aprendemos como esses algoritmos fazem a abstração da rede de computadores em um grafo com nós e enlaces. Com essa abstração, podemos pesquisar a rica teoria do roteamento de caminho mais curto em grafos, que foi desenvolvida nos últimos 40 anos nas comunidades de pesquisa operacional e de algoritmos. Vimos que há duas abordagens amplas: uma centralizada (global), em que cada nó obtém o mapeamento completo da rede e aplica independentemente o algoritmo de roteamento de caminho mais curto, e uma descentralizada, em que nós individuais têm apenas um quadro parcial da rede inteira e, mesmo assim, trabalham em conjunto para entregar pacotes ao longo das rotas mais curtas. Também estudamos como a hierarquia é utilizada para lidar com o problema da escala, dividindo redes

de grande porte em domínios administrativos independentes denominados sistemas autônomos (ASs). Cada AS direciona independentemente seus datagramas pelo sistema, assim como cada país distribui correspondência postal pelo seu território. Aprendemos como abordagens centralizadas, descentralizadas e hierárquicas estão incorporadas nos principais protocolos de roteamento da Internet: RIP, OSPF e BGP. Concluímos nosso estudo de algoritmos de roteamento considerando o roteamento por difusão (*broadcast*) e para um grupo (*multicast*).

Agora que concluímos nosso estudo da camada de rede, nossa jornada segue rumo a um nível mais baixo da pilha de protocolos, isto é, à camada de enlace. Como a camada de rede, a camada de enlace é também parte do núcleo da rede. Mas veremos no próximo capítulo que a camada de enlace tem a tarefa muito mais localizada de movimentar pacotes entre nós no mesmo enlace ou rede local. Embora essa tarefa possa à primeira vista parecer trivial quando comparada às tarefas da camada de rede, veremos que a camada de enlace envolve uma série de questões importantes e fascinantes que podem nos manter ocupados por muito tempo.

EXERCÍCIOS DE FIXAÇÃO E PERGUNTAS

Questões de revisão do Capítulo 4

SEÇÕES 4.1–4.2

R1. Vamos rever um pouco da terminologia usada neste livro. Lembre-se de que o nome de um pacote na camada de transporte é *segmento* e que o nome de um pacote na camada de enlace é *quadro*. Qual é o nome de um pacote de camada de rede? Lembre-se de que roteadores e comutadores da camada de enlace são denominados *comutadores de pacotes*. Qual é a diferença fundamental entre um roteador e um comutador da camada de enlace? Lembre-se de que usamos o termo *roteadores* tanto para redes de datagramas quanto para redes de CVs.

R2. Quais são as duas funções mais importantes de camada de rede em uma rede de datagramas? Quais são as três funções mais importantes de camada de rede em uma rede com circuitos virtuais?

R3. Qual é a diferença entre rotear e repassar?

R4. Os roteadores nas redes de datagramas e nas redes de circuitos virtuais usam tabelas de repasse? Caso usem, descreva as tabelas de repasse para ambas as classes de redes.

R5. Descreva alguns serviços hipotéticos que a camada de rede poderia oferecer a um pacote individual. Faça o mesmo para um fluxo de pacotes. Alguns dos serviços hipotéticos que você descreveu são fornecidos pela camada de rede da Internet? Alguns são fornecidos pelo modelo de serviço ATM CBR? Alguns são fornecidos pelo modelo de serviço ATM ABR?

R6. Cite algumas aplicações que poderiam se beneficiar do modelo de serviço ATM CBR.

SEÇÃO 4.3

R7. Discuta por que cada porta de entrada em um roteador de alta velocidade armazena uma cópia de sombra da tabela de repasse.

R8. Três tipos de elementos de comutação são discutidos na Seção 4.3. Cite e descreva brevemente cada tipo. Qual (se houver algum) pode enviar múltiplos pacotes em paralelo pelo elemento?

R9. Descreva como pode ocorrer perda de pacotes em portas de entrada. Descreva como a perda de pacotes pode ser eliminada em portas de entrada (sem usar buffers infinitos).

R10. Descreva como pode ocorrer perda de pacotes em portas de saída. Essa perda poderia ser impedida aumentando a velocidade de fábrica do comutador?

R11. O que é bloqueio HOL? Ele ocorre em portas de saída ou em portas de entrada?

SEÇÃO 4.4

R12. Roteadores têm endereços IP? Em caso positivo, quantos?

R13. Qual é o equivalente binário de 32 bits para o endereço IP 223.1.3.27?

R14. Visite um hospedeiro que usa DHCP para obter seu endereço IP, máscara de rede, roteador de *default* e endereço IP de seu servidor DNS local. Faça uma lista desses valores.

R15. Suponha que haja três roteadores entre os hospedeiros de origem e de destino. Ignorando a fragmentação, um datagrama IP enviado do hospedeiro de origem até o hospedeiro de destino transitará por quantas interfaces? Quantas tabelas de repasse serão indexadas para deslocar o datagrama desde a origem até o destino?

R16. Suponha que uma aplicação gere blocos de 40 bytes de dados a cada 20 ms e que cada bloco seja encapsulado em um segmento TCP e, em seguida, em um datagrama IP. Que porcentagem de cada datagrama será sobrecarga e que porcentagem será dados de aplicação?

R17. Suponha que o hospedeiro A envie ao hospedeiro B um segmento TCP encapsulado em um datagrama IP. Quando o hospedeiro B recebe o datagrama, como sua camada de rede sabe que deve passar o segmento (isto é, a carga útil do datagrama) para TCP e não para UDP ou qualquer outra coisa?

R18. Suponha que você compre um roteador sem fio e o conecte a seu modem a cabo. Suponha também que seu ISP designe dinamicamente um endereço IP a seu dispositivo conectado (isto é, seu roteador sem fio). Suponha ainda que você tenha cinco PCs em casa e que usa 802.11 para conectá-los sem fio ao roteador. Como são designados endereços IP aos cinco PCs? O roteador sem fio usa NAT? Por quê?

R19. Compare os campos de cabeçalho do IPv4 e do IPv6 e aponte suas diferenças. Eles têm algum campo em comum?

R20. Afirma-se que, quando o IPv6 implementa túneis via roteadores IPv4, o IPv6 trata os túneis IPv4 como protocolos de camada de enlace. Você concorda com essa afirmação? Explique sua resposta.

SEÇÃO 4.5

R21. Compare e aponte as diferenças entre os algoritmos de roteamento de estado de enlace e por vetor de distâncias.

R22. Discuta como a organização hierárquica da Internet possibilitou estender seu alcance para milhões de usuários.

R23. É necessário que todo sistema autônomo use o mesmo algoritmo de roteamento intra-AS? Justifique sua resposta.

SEÇÃO 4.6

R24. Considere a Figura 4.37. Começando com a tabela original em *D*, suponha que *D* receba de *A* o seguinte anúncio:

Sub-rede de destino	Roteador seguinte	Número de saltos até o destino
z	C	10
w	—	1
x	—	1
....	....	....

A tabela em *D* mudará? Em caso afirmativo, como?

R25. Compare os anúncios utilizados por RIP e OSPF e aponte suas diferenças.

R26. Complete: anúncios RIP em geral anunciam o número de saltos até vários destinos. Atualizações BGP, por outro lado, anunciam ________________ aos diversos destinos.

R27. Por que são usados protocolos inter-AS e intra-AS diferentes na Internet?

R28. Por que considerações políticas são tão importantes para protocolos intra-AS, como o OSPF e o RIP, quanto para um protocolo de roteamento inter-AS, como BGP?

R29 Defina e aponte as diferenças entre os seguintes termos: *sub-rede*, *prefixo* e *rota BGP*.

R30. Como o BGP usa o atributo NEXT-HOP? Como ele usa o atributo AS-PATH?

R31. Descreva como um administrador de rede de um ISP de nível superior pode executar uma política ao configurar o BGP.

SEÇÃO 4.7

R32. Cite uma diferença importante entre a execução da abstração de difusão por meio de múltiplas transmissões individuais e a de uma única difusão com suporte da rede (roteador).

R33. Para cada uma das três abordagens gerais que estudamos para a comunicação por difusão (inundação não controlada, inundação controlada e difusão por *spanning tree*), as seguintes declarações são verdadeiras ou falsas? Você pode considerar que não há perda de pacotes por estouro de buffers e que todos os pacotes são entregues em um enlace na ordem em que foram enviados.

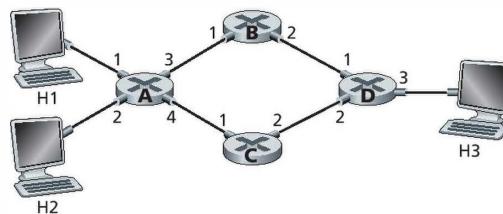
a. Um nó pode receber várias cópias do mesmo pacote.

b. Um nó pode repassar várias cópias de um pacote pelo mesmo enlace de saída.

R34. Quando um hospedeiro se junta a um grupo, ele deve mudar seu endereço IP para o endereço do grupo ao qual está se juntando?

R35. Quais são os papéis desempenhados pelo protocolo IGMP e por um protocolo de roteamento para um grupo de longa distância?

R36. Qual é a diferença entre uma árvore compartilhada por um grupo e uma árvore de origem no contexto do roteamento para um grupo?

PROBLEMAS

P1. Nesta questão, consideramos alguns dos prós e dos contras de redes decircuitos virtuais e redes de datagramas.

a. Suponha que roteadores foram submetidos a condições que poderiam levá-los a falhar com muita frequência. Isso seria um argumento em favor de um CV ou arquitetura de datagrama? Por quê?

b. Suponha que um nó de origem e um de destino solicitem que uma quantidade fixa de capacidade esteja sempre disponível em todos os roteadores no caminho entre o nó de origem e de destino, para o uso exclusivo de fluxo de tráfego entre esse nós. Essas ações favorecem uma arquitetura de circuitos virtuais ou de datagramas? Por quê?

c. Suponha que os enlaces e os roteadores da rede nunca falhem e que os caminhos de roteamento usados entre as duplas de origem/destino permaneçam constantes. Nesse cenário, a arquitetura de circuitos virtuais ou de datagramas possui mais sobrecarga de tráfego de controle? Por quê?

P2. Considere uma rede de circuitos virtuais. Suponha que o número do CV é um campo de 8 bits.

a. Qual é o número máximo de circuitos virtuais que pode ser transportado por um enlace?

b. Suponha que um nó central determine caminhos e números de CVs no estabelecimento da conexão. Suponha que o mesmo número de CV seja usado em cada enlace no caminho do CV. Descreva como o nó central poderia determinar o número do CV no estabelecimento da conexão. É possível haver um número de CVs ativos menor do que o máximo determinado na parte (a) e, mesmo assim, não haver nenhum número de CV livre em comum?

c. Suponha que sejam permitidos números diferentes de CVs em cada enlace no caminho de um CV. Durante o estabelecimento da conexão, após a determinação de um caminho fim a fim, descreva como

os enlaces podem escolher seus números de CVs e configurar suas tabelas de repasse de uma maneira descentralizada, sem depender de um nó central.

P3. Uma tabela de repasse bem básica em uma rede de CVs tem quatro colunas. O que significam os valores em cada uma dessas colunas? Uma tabela de repasse bem básica em uma rede de datagramas tem duas colunas. O que significam os valores em cada coluna?

P4. Considere a rede a seguir.

a. Suponha que seja uma rede de datagramas. Mostre a tabela de repasse no roteador A, de modo que todo o tráfego destinado ao hospedeiro H3 seja encaminhado pela interface 3.

b. Suponha que esta rede seja uma rede de datagramas. Você consegue compor uma tabela de repasse no roteador A, de modo que todo o tráfego de H1 destinado ao hospedeiro H3 seja encaminhado pela interface 3, enquanto todo o tráfego de H2 destinado ao hospedeiro H3 seja encaminhado pela interface 4? (*Dica:* esta é uma pergunta capciosa.)

c. Suponha, agora, que esta rede seja uma rede de circuitos virtuais e que haja uma chamada em andamento entre H1 e H3, e outra chamada em andamento entre H2 e H3. Elabore uma tabela de repasse no roteador A, de modo que todo o tráfego de H1 destinado ao hospedeiro H3 seja encaminhado pela interface 3, enquanto todo o tráfego de H2 destinado ao hospedeiro H3 seja encaminhado pela interface 4.

d. Admitindo o mesmo cenário de (c), elabore tabelas de repasse nos nós B, C e D.

P5. Considere uma rede de circuito virtual cujo campo de número de CV tenha 2 bits. Suponha que a rede queira estabelecer um circuito virtual por quatro enlaces: A, B, C e D. Imagine que, nesse momento, cada enlace esteja carregando dois outros circuitos virtuais e que os números desses outros CVs são os seguintes:

Enlace A	Enlace B	Enlace C	Enlace D
00	01	10	11
01	10	11	00

Ao responder às perguntas a seguir, tenha em mente que cada um dos CVs pode estar atravessando apenas um dos quatro enlaces.

a. Se cada CV tiver de usar o mesmo número de CV em todos os enlaces ao longo de seu caminho, qual número de CV poderia ser designado ao novo CV?

b. Se fosse permitido que cada CV tivesse um número de CV diferente nos diferentes enlaces em seu caminho (de modo que a tabela de repasse tenha de realizar tradução de número de CV), quantas combinações diferentes de quatro números de CVs (um para cada enlace) poderiam ser usadas?

P6. No texto usamos o termo *serviço orientado para conexão* para descrever a camada de transporte e *serviço de conexão* para a camada de rede. Por que essa sutileza na terminologia?

P7. Suponha que dois pacotes cheguem a duas portas de entrada diferentes de um roteador exatamente ao mesmo tempo. Suponha também que não haja outros pacotes em lugar algum no roteador.

a. Suponha que os dois pacotes devam ser repassados a duas portas de saída *diferentes*. É possível repassar os dois pacotes pelo elemento de comutação ao mesmo tempo quando o elemento usa um *barramento compartilhado*?

b. Imagine que os dois pacotes devam ser repassados a duas portas de saída *diferentes*. É possível repassar os dois pacotes pelo elemento de comutação ao mesmo tempo quando o elemento usa o tipo *crossbar*?

c. Considere que os dois pacotes devam ser repassados para a *mesma* porta de saída. É possível repassar os dois pacotes pelo elemento de comutação ao mesmo tempo quando o elemento usa um tipo *crossbar*?

P8. Na Seção 4.3, observamos que o atraso máximo de fila é $(n-1)D$ se o elemento de comutação for n vezes mais rápido do que as taxas das linhas de entrada. Suponha que todos os pacotes tenham o mesmo comprimento, n pacotes chegam ao mesmo tempo às n portas de entrada e todos os n pacotes querem ser encaminhados para *diferentes* portas de saída. Qual é o atraso máximo para um pacote para (a) a memória, (b) o barramento e (c) os elementos de comutação do tipo *crossbar*?

P9. Considere o comutador a seguir. Suponha que todos os datagramas possuam o mesmo comprimento, que o comutador opere de uma maneira segmentada e síncrona, e que em um intervalo de tempo (*time slot*) um datagrama possa ser transferido de uma porta de entrada para uma porta de saída. A malha de comutação é um *crossbar* no qual, no máximo, um datagrama possa ser transferido para uma determinada porta de saída em um intervalo de tempo, mas portas de saída diferentes podem receber datagramas de portas de entrada diferentes em um único intervalo de tempo. Qual é o número mínimo de intervalos de tempo necessário para transferir os pacotes mostrados das portas de entrada para suas portas de saída, admitindo qualquer ordem de escalonamento de fila que você quiser (ou seja, não é necessário o bloqueio HOL)? Qual é o maior número de intervalos necessários, admitindo uma ordem de escalonamento de pior caso e que uma fila de entrada não vazia nunca fica ociosa?

P10. Considere uma rede de datagramas que usa endereços de hospedeiro de 32 bits. Suponha que um roteador tenha quatro enlaces, numerados de 0 a 3, e que os pacotes têm de ser repassados para as interfaces de enlaces desta forma:

Faixa do endereço de destino	Interface de enlace
11100000 00000000 00000000 00000000 até 11100000 00111111 11111111 11111111	0
11100000 01000000 00000000 00000000 até 11100000 01000000 11111111 11111111	1
11100000 01000001 00000000 00000000 até 11100001 01111111 11111111 11111111	2
senão	3

a. Elabore uma tabela de repasse que tenha cinco registros, use correspondência do prefixo mais longo e repasse pacotes para as interfaces de enlace corretas.

b. Descreva como sua tabela de repasse determina a interface de enlace apropriada para datagramas com os seguintes endereços:

11001000 10010001 01010001 01010101
11100001 01000000 11000011 00111100
11100001 10000000 00010001 01110111

P11. Considere uma rede de datagramas que utiliza endereços de hospedeiro de 8 bits. Suponha que um roteador utilize a correspondência do prefixo mais longo e tenha a seguinte tabela de repasse:

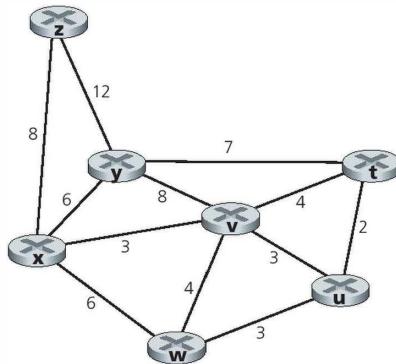
Prefixo a comparar	Interface
00	0
010	1
011	2
10	2
11	3

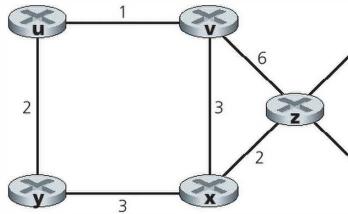
Para cada uma das quatro interfaces, forneça a faixa associada de endereços de hospedeiro de destino e o número de endereços na faixa.

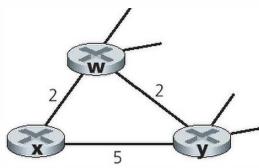
P12. Considere uma rede de datagramas que usa endereços de hospedeiros de 8 bits. Suponha que um roteador use a correspondência do prefixo mais longo e tenha a seguinte tabela de repasse:

Prefixo a comparar	Interface
1	0
10	1
111	2
senão	3

Para cada uma das quatro interfaces, forneça a faixa associada de endereços de hospedeiro de destino e o número de endereços na faixa.

P13. Considere um roteador que interconecta três sub-redes: 1, 2 e 3. Suponha que todas as interfaces de cada uma dessas três sub-redes tenha de ter o prefixo 223.1.17/24. Suponha também que a sub-rede 1 tenha de suportar até 60 interfaces, a sub-rede 2 tenha de suportar até 90 interfaces e a sub-rede 3, 12 interfaces. Dê três endereços de rede (da forma a.b.c.d/x) que satisfaçam essas limitações.

P14. Na Seção 4.2.2 é dado um exemplo de tabela de repasse (usando a correspondência de prefixo mais longo). Reescreva a tabela usando a notação a.b.c.d/x em vez da notação de cadeia binária.

P15. No Problema P10, solicitamos que você elaborasse uma tabela de repasse (usando a correspondência de prefixo mais longo). Reescreva a tabela usando a notação a.b.c.d/x em vez da notação de cadeia binária.

P16. Considere uma sub-rede com prefixo 128.119.40.128/26. Dê um exemplo de um endereço IP (na forma xxx.xxx.xxx.xxx) que possa ser designado para essa rede. Suponha que um ISP possua o bloco de endereços na forma 128.119.40.64/26. Suponha que ele queira criar quatro sub-redes a partir desse bloco, e que cada bloco tenha o mesmo número de endereços IP. Quais são os prefixos (na forma a.b.c.d/x) para as quatro sub-redes?

P17. Considere a topologia mostrada na Figura 4.17. Denomine as três sub-redes com hospedeiros (começando em sentido horário, a partir da posição das 12h) como A, B e C. Denomine as sub-redes sem hospedeiros como D, E e F.

a. Designe endereços de rede a cada uma das seis sub-redes, com as seguintes restrições: todos os endereços deverão ser alocados a partir de 214.97.254/23; a sub-rede A deve ter endereços suficientes para suportar 250 interfaces; a sub-rede B deve ter endereços suficientes para suportar 120 interfaces e a sub-rede C deve ter endereços suficientes para suportar 120 interfaces. É claro que cada uma das sub-redes D, E e F deve poder suportar duas interfaces. Para cada sub-rede, a designação deve tomar a forma a.b.c.d/x ou a.b.c.d/x – e.f.g.h/y.

b. Usando a resposta dada no item (a), elabore as tabelas de repasse (usando a correspondência de prefixo mais longo) para cada um dos três roteadores.

P18. Use o serviço whois no American Registry for Internet Numbers (http://www.arin.net/whois) para determinar os blocos de endereço IP para três universidades. Os serviços whois podem ser usados para determinar com certeza o local geográfico de um endereço IP específico? Use www.maxmind.com para determinar os locais dos servidores Web em cada universidade.

P19. Considere enviar um datagrama de 2.400 bytes por um enlace que tem uma MTU de 700 bytes. Suponha que o datagrama original esteja marcado com o número de identificação 422. Quantos fragmentos são gerados? Quais são os valores em vários campos dos datagramas IP gerados em relação à fragmentação?

P20. Suponha que entre o hospedeiro de origem A e o hospedeiro destinatário B os datagramas estejam limitados a 1.500 bytes (incluindo cabeçalho). Admitindo um cabeçalho IP de 20 bytes, quantos datagramas seriam necessários para enviar um arquivo MP3 de 5 milhões de bytes? Explique como você obteve a resposta.

P21. Considere a configuração de rede da Figura 4.22. Suponha que o ISP designe ao roteador o endereço 24.34.112.235 e que o endereço da rede residencial seja 192.168.1/24.

a. Designe endereços a todas as interfaces na rede residencial.

b. Suponha que haja duas conexões TCP em curso em cada hospedeiro, todas para a porta 80 no hospedeiro 128.119.40.86. Forneça os seis registros correspondentes na tabela de tradução NAT.

P22. Suponha que você esteja interessado em detectar o número de hospedeiros por trás da NAT. Você observa que a camada IP traz um número de identificação, de modo sequencial, em cada pacote IP. O número de identificação do primeiro pacote IP gerado por um hospedeiro é aleatório, e os números de identificação subsequentes são determinados sequencialmente. Admita que todos os pacotes IP gerados por hospedeiros por trás da NAT sejam enviados para o mundo exterior.

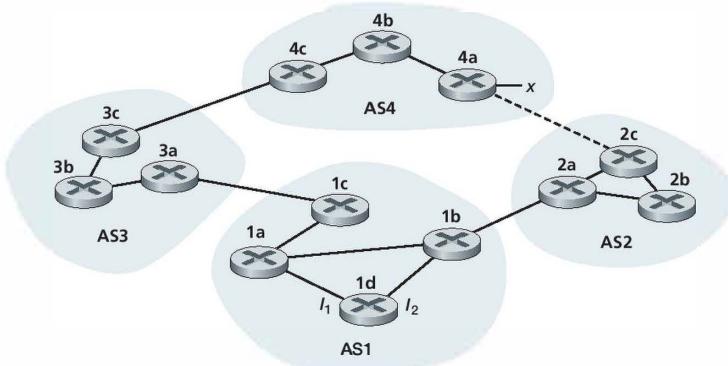
a. Com base nessa observação e admitindo que você pode analisar todos os pacotes enviados para fora pela NAT, você pode descrever uma técnica simples que detecte o número de hospedeiros únicos por trás da NAT? Justifique sua resposta.

b. Se os números de identificação não são determinados de maneira sequencial, e sim aleatória, sua técnica funcionaria? Justifique sua resposta.

P23. Neste problema estudaremos o impacto das NATs sobre aplicações P2P. Suponha que um parceiro com nome de usuário Arnold descubra, por meio de consulta, que um parceiro com nome de hospedeiro Bernard tem um arquivo que ele, Arnold, quer descarregar. Suponha também que Bernard e Arnold estejam por trás de uma NAT. Tente elaborar uma técnica que permita a Arnold estabelecer uma conexão TCP com Bernard sem a configuração da NAT específica da aplicação. Se você tiver dificuldade na elaboração dessa técnica, discuta o motivo.

P24. Considerando a Figura 4.27, enumere os caminhos de y a u que não tenham laços.

P25. Repita o Problema P24 considerando os caminhos de x a z, z a u e z a w.

P26. Considere a seguinte rede. Com os custos de enlace indicados, use o algoritmo do caminho mais curto de Dijkstra para calcular o caminho mais curto de x até todos os nós da rede. Mostre como o algoritmo funciona calculando uma tabela semelhante à Tabela 4.3.

P27. Considere a rede mostrada no Problema P26. Usando o algoritmo de Dijkstra e mostrando seu trabalho usando uma tabela semelhante à Tabela 4.3, faça o seguinte:

a. Calcule o caminho mais curto de *t* até todos os nós da rede.

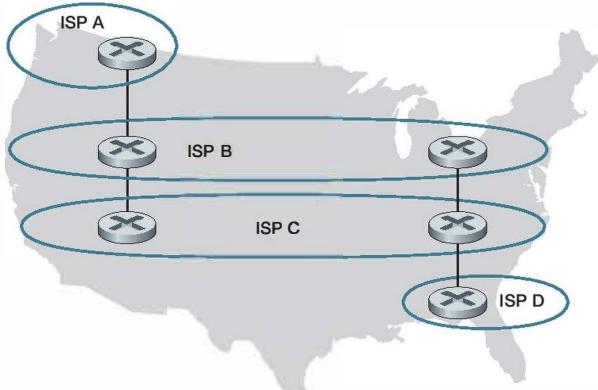
b. Calcule o caminho mais curto de *u* até todos os nós da rede.

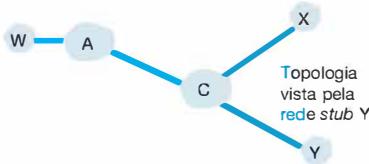
c. Calcule o caminho mais curto de *v* até todos os nós da rede.

d. Calcule o caminho mais curto de *w* até todos os nós da rede.

e. Calcule o caminho mais curto de *y* até todos os nós da rede.

f. Calcule o caminho mais curto de *z* até todos os nós da rede.

P28. Considere a rede mostrada a seguir e admita que cada nó inicialmente conheça os custos até cada um de seus vizinhos. Considere o algoritmo de vetor de distâncias e mostre os registros na tabela de distâncias para o nó *z*.

P29. Considere uma topologia geral (isto é, não a rede específica mostrada antes) e uma versão síncrona do algoritmo de vetor de distâncias. Suponha que, a cada iteração, um nó troque seus vetores de distâncias com seus vizinhos e receba os vetores de distâncias deles. Supondo que o algoritmo comece com cada nó conhecendo apenas os custos até seus vizinhos imediatos, qual é o número máximo de iterações exigidas até que o algoritmo distribuído convirja? Justifique sua resposta.

P30. Considere o fragmento de rede mostrado a seguir. *x* tem apenas dois vizinhos ligados a ele: *w* e *y*. *w* tem um caminho de custo mínimo até o destino *u* (não mostrado) de 5 e *y* tem um caminho de custo mínimo *u* de 6. Os caminhos completos de *w* e de *y* até *u* (e entre *w* e *y*) não são mostrados. Todos os valores dos custos de enlace na rede são números inteiros estritamente positivos.

a. Dê os vetores de distâncias de x para os destinos w, y e u.

b. Dê uma mudança de custo de enlace para $c(x, w)$ ou para $c(x, y)$ tal que x informará a seus vizinhos um novo caminho de custo mínimo até u como resultado da execução do algoritmo de vetor de distâncias.

c. Dê uma mudança de custo de enlace para $c(x, w)$ ou para $c(x, y)$ tal que x não informará a seus vizinhos um novo caminho de custo mínimo até u como resultado da execução do algoritmo de vetor de distâncias.

P31. Considere a topologia de três nós mostrada na Figura 4.30. Em vez de ter os custos de enlace da Figura 4.30, os custos de enlace são: $c(x, y) = 3$, $c(y, z) = 6$, $c(z, x) = 4$. Calcule as tabelas de distâncias após a etapa de inicialização e após cada iteração de uma versão síncrona do algoritmo de vetor de distâncias (como fizemos em nossa discussão anterior da Figura 4.30).

P32. Considere o problema da contagem até o infinito no roteamento de vetor de distâncias. Esse problema ocorrerá se reduzirmos o custo de um enlace? Por quê? E se conectarmos dois nós que não possuem enlace?

P33. Demonstre que, para o algoritmo de vetor de distâncias na Figura 4.30, cada valor no vetor de distâncias $D(x)$ é não crescente e, consequentemente, se estabilizará em um número finito de etapas.

P34. Considere a Figura 4.31. Suponha que haja outro roteador, w, conectado aos roteadores y e z. Os custos de todos os enlaces são: $c(x, y) = 4$, $c(x, z) = 50$, $c(y, w) = 1$, $c(z, w) = 1$, $c(y, z) = 3$. Suponha que a reversão envenenada seja utilizada no algoritmo de roteamento de vetor de distâncias.

a. Quando o roteamento de vetor de distâncias é estabilizado, os roteadores w, y e z informam uns aos outros sobre suas distâncias para x. Que valores de distância eles podem informar uns aos outros?

b. Agora suponha que o custo do enlace entre x e y aumente para 60. Haverá um problema de contagem até o infinito mesmo se a reversão envenenada for utilizada? Por quê? Se houver um problema de contagem até o infinito, então quantas iterações são necessárias para o roteamento de vetor de distâncias alcançar um estágio estável novamente? Justifique sua resposta.

c. Como você modifica $c(y,z)$ de modo que não haja qualquer problema de contagem até o infinito se $c(y,x)$ mudar de 4 para 60?

P35. Descreva como laços nos caminhos podem ser detectados com BGP.

P36. Um roteador BGP sempre escolherá uma rota sem laços com o menor comprimento de AS-PATH? Justifique sua resposta.

P37. Considere a rede a seguir. Suponha que AS3 e AS2 estejam rodando o OSPF para seu protocolo de roteamento intra-AS. Suponha que AS1 e AS4 estejam rodando o RIP para seu protocolo de roteamento intra-AS. Suponha que o eBGP e o iBGP sejam usados para o protocolo de roteamento intra-AS. Inicialmente, suponha que *não* haja enlace físico entre AS2 e AS4.

a. O roteador 3c sabe sobre o prefixo x por qual protocolo de roteamento: OSPF, RIP, eBGP ou iBGP?

b. O roteador 3a sabe sobre o prefixo x por qual protocolo de roteamento?

c. O roteador 1c sabe sobre o prefixo x por qual protocolo de roteamento?

d. O roteador 1d sabe sobre o prefixo x por qual protocolo de roteamento?

P38. Referindo-se ao problema anterior, uma vez que o roteador 1d sabe sobre *x*, ele inserirá uma entrada (*x*, *I*) em sua tabela de repasse.

a. *I* será igual a I_1 ou I_2 para essa entrada? Justifique a resposta em uma frase.

b. Agora suponha que haja um enlace físico entre AS2 e AS4, ilustrado pela linha pontilhada. Suponha que o roteador 1d saiba que *x* é acessível por meio de AS2 e de AS3. *I* será definido para I_1 ou I_2? Justifique a resposta em uma frase.

c. Agora suponha que haja outro AS, denominado AS5, que fica no caminho entre AS2 e AS4 (não ilustrado no diagrama). Suponha que o roteador 1d saiba que *x* é acessível por meio de AS2 AS5 AS4, bem como de AS3 AS4. *I* será definido para I_1 ou I_2? Em uma frase, explique o motivo.

P39. Considere a rede a seguir. O ISP B provê serviço nacional de *backbone* ao ISP regional A. O ISP C provê serviço nacional de *backbone* ao ISP regional D. Cada ISP consiste em um AS. Usando BGP, B e C se emparelham em dois lugares. Considere o tráfego na direção de A a D. B preferiria passar esse tráfego para C na Costa Oeste (de modo que C teria de absorver o custo de transportar o tráfego através do país), enquanto C preferiria receber o tráfego via seu ponto de emparelhamento com B na Costa Leste (de modo que B transportaria o tráfego através do país). Qual mecanismo BGP poderia ser usado por C de modo que B entregasse o tráfego de A a D em seu ponto de emparelhamento na Costa Leste? Para responder a essa pergunta, você precisará estudar muito bem a especificação do BGP.

P40. Na Figura 4.42, considere a informação de caminho que chega às sub-redes *stub* W, X e Y. Com base na informação disponível em W e X, quais são as respectivas visões da topologia da rede? Justifique sua resposta. A topologia vista de Y é mostrada a seguir.

P41. Considere a Figura 4.42. B nunca encaminharia tráfego destinado a Y via X com base no roteamento BGP. Mas existem muitas aplicações conhecidas para as quais os pacotes de dados vão primeiro para X e depois fluem para Y. Identifique tal aplicação e descreva como os pacotes de dados percorrem um caminho não determinado pelo roteamento BGP.

P42. Na Figura 4.42, suponha que haja outra rede *stub* V que seja cliente do ISP A. Suponha que B e C tenham uma relação de emparelhamento, e que A seja cliente de B e de C. Suponha, ainda, que A gostaria de ter o tráfego destinado para W vindo apenas de B, e o tráfego destinado para V vindo de B ou C. Como A deveria anunciar suas rotas para B e C? Quais rotas AS são recebidas por C?

P.43. Suponha que os ASs X e Z não estejam conectados diretamente, mas estejam conectados pelo AS Y. Suponha ainda que X tenha um acordo de emparelhamento com Y, e que Y tenha um acordo de emparelhamento com Z. Por fim, suponha que Z queira transitar todo o tráfego de Y, mas não queira transitar o tráfego de X. O BGP permite que Z execute essa política?

P44. Considere a rede de sete nós (com nós rotulados de *t* a *z*) do Problema 26. Mostre a árvore de custo mínimo com raiz em *z* que inclua (como hospedeiros finais) os nós *u*, *v*, *w* e *y*. Justifique informalmente por que sua árvore é uma árvore de custo mínimo.

P45. Considere as duas abordagens básicas identificadas para fazer a difusão: emulação de transmissão individual e difusão de camada de rede (isto é, assistido por roteador) e suponha que seja usada difusão de *spanning tree* para fazer difusão de camada de rede. Considere um único remetente e 32 destinatários. Suponha que o remetente esteja conectado aos destinatários por uma árvore binária de roteadores. Qual é o custo para enviar um pacote por difusão nos casos da emulação de transmissão individual e de difusão de camada de rede para essa topologia? Aqui, cada vez que um pacote (ou cópia de um pacote) é enviado por um único enlace, ele incorre em uma unidade de custo. Qual topologia para interconectar o remetente, os destinatários e os roteadores fará que os custos da emulação individual e de difusão verdadeira da camada de rede fiquem o mais longe possível um do outro? Você pode escolher quantos roteadores quiser.

P46. Considere a operação do algoritmo de repasse de caminho inverso (RPF) na Figura 4.44. Usando a mesma topologia, descubra um conjunto de caminhos de todos os nós até o nó de origem *A* (e indique esses caminhos em um grafo usando linhas grossas como as da Figura 4.44) de modo que, se esses caminhos forem os de menor custo, então o nó *B* receberia uma cópia das mensagens de difusão de *A* dos nós *A*, *C* e *D* sob RPF.

P47. Considere a topologia ilustrada na Figura 4.44. Suponha que todos os enlaces tenham custo unitário e que o nó *E* é a origem da difusão. Usando setas como as mostradas na Figura 4.44, indique enlaces pelos quais pacotes serão repassados usando RPF e enlaces pelos quais pacotes não serão repassados, dado que o nó *E* é a origem.

P48. Repita o Problema P47 utilizando o gráfico do Problema P26. Suponha que *z* seja a origem da difusão e que os custos do enlace são aqueles mostrados no Problema P26.

P49. Considere a topologia mostrada na Figura 4.46 e suponha que cada enlace tenha custo unitário. Suponha que o nó *C* seja escolhido como o centro de um algoritmo de roteamento para um grupo baseado em centro. Supondo que cada roteador conectado use seu caminho de menor custo até o nó *C* para enviar mensagens de

adesão a *C*, desenhe a árvore de roteamento baseada no centro resultante. É uma árvore de custo mínimo? Justifique sua resposta.

P50. Repita o Problema P49, usando o gráfico do Problema P26. Admita que o nó central seja *v*.

P51. Na Seção 4.5.1 estudamos o algoritmo de roteamento de estado de enlace de Dijkstra para calcular os caminhos individuais que são os de menor custo da origem até todos os destinos. Poderíamos imaginar que a união desses caminhos forme uma **árvore individual de caminho de menor custo** (ou uma árvore individual de caminho mais curto, se todos os custos de enlaces fossem idênticos). Construindo um contraexemplo (um exemplo que nega essa afirmação), mostre que a árvore de caminho de menor custo *nem sempre* é a mesma que uma *spanning tree* mínima.

P52. Considere uma rede na qual todos os nós estão conectados a três outros nós. Em uma única fase de tempo, um nó pode receber de seus vizinhos todos os pacotes transmitidos por difusão, duplicar os pacotes e enviá-los a todos os seus vizinhos (exceto ao nó que enviou um dado pacote). Na próxima fase, nós vizinhos podem receber, duplicar e repassar esses pacotes e assim por diante. Suponha que seja utilizada a inundação não controlada para prover difusão a essa rede. Na fase de tempo *t*, quantas cópias do pacote *broadcast* serão transmitidas, admitindo que, durante a fase 1, um único pacote *broadcast* é transmitido pelo nó de origem a seus três vizinhos?

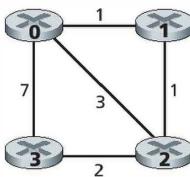
P53. Vimos na Seção 4.7 que não há um protocolo de camada de rede que possa ser usado para identificar os hospedeiros que participam de um grupo (*multicast*). Isto posto, como aplicações de transmissão para um grupo podem aprender as identidades dos hospedeiros que estão participando de um grupo?

P54. Projete (dê uma descrição em pseudocódigo) um protocolo de nível de aplicação que mantenha os endereços de hospedeiros para todos os hospedeiros participantes de um grupo (*multicast*). Identifique o serviço de rede (individual ou para um grupo) que é usado por seu protocolo e indique se seu protocolo está enviando mensagens dentro ou fora da banda (com relação ao fluxo de dados de aplicação entre os participantes do grupo) e por quê.

P55. Qual é o tamanho do espaço de endereços de grupo? Suponha agora que dois grupos escolham aleatoriamente um endereço de grupo. Qual é a probabilidade de que escolham o mesmo endereço? Suponha agora que mil grupos estejam em operação ao mesmo tempo e escolham seus endereços de grupo aleatoriamente. Qual é a probabilidade de que uns interfiram nos outros?

TAREFAS DE PROGRAMAÇÃO DE *SOCKETS*

Ao final do Capítulo 2, existem quatro tarefas de programação de *sockets*. A seguir, você verá uma quinta tarefa que emprega ICMP, um protocolo discutido neste capítulo.

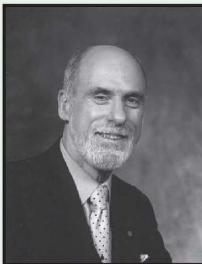
Tarefa 5: ICMP ping

Ping é uma aplicação popular para redes, usada para testar, de um local remoto, se determinado hospedeiro está ativo e pode ser alcançado. Em geral é usada para medir a latência entre o hospedeiro cliente e o hospedeiro de destino. Ele funciona enviando pacotes ICMP de "requisição de eco" (isto é, pacotes *ping*) ao hospedeiro de destino e escutando as "respostas de eco" ICMP (isto é, pacotes *pong*). Ping mede o RTT, registra perda de pacote e calcula um resumo estatístico de diversas trocas *ping-pong* (o mínimo, média, máximo e desvio-padrão dos tempos de viagem de ida e volta).

Neste laboratório, você escreverá sua própria aplicação Ping em Python. Sua aplicação usará ICMP. Mas, para simplificar seu programa, você não seguirá exatamente a especificação oficial no RFC 1739. Observe que só precisará escrever o lado cliente do programa, pois a funcionalidade necessária no lado servidor já está embutida em todos os sistemas operacionais. Você poderá achar todos os detalhes desta tarefa, bem como trechos importantes do código em Python, no site de apoio do livro.

TAREFAS DE PROGRAMAÇÃO

Nesta tarefa de programação, você escreverá um conjunto "distribuído" de procedimentos que executam um roteamento de vetor de distâncias assíncrono distribuído para a rede mostrada a seguir.

Você deve escrever as seguintes rotinas que "rodarão" assincronamente dentro do ambiente emulado fornecido para essa tarefa. Para o nó 0, escreverá estas rotinas:

- *rtinit0()*. Essa rotina será chamada uma vez no início da emulação. *rtinit0()* não tem argumentos. Você deve inicializar sua tabela de distâncias no nó 0 para refletir os custos diretos de 1, 3 e 7 até os nós 1, 2 e 3, respectivamente. Na figura anterior, todos os enlaces são bidirecionais e os custos em ambas as direções são idênticos. Após inicializar a tabela de distâncias e quaisquer outras estruturas de dados necessárias às rotinas de seu nó 0, este deve então enviar a seus vizinhos diretamente ligados (nesse caso, 1, 2 e 3) o custo de seus caminhos de custo mínimo para todos os outros nós da rede. Essa informação do custo mínimo é enviada aos nós vizinhos em um pacote de atualização de roteamento chamando a rotina *tolayer2()*, conforme descrito na tarefa completa. O formato do pacote de atualização de roteamento também está descrito na tarefa completa.
- *rtupdate0(struct rtpkt *rcvdpkt)*. Essa rotina será chamada quando o nó 0 receber um pacote de roteamento que foi enviado a ele por um de seus vizinhos diretamente conectados. O parâmetro **rcvdpkt* é um indicador para o pacote que foi recebido. *rtupdate0()* é o "coração" do algoritmo de vetor de distâncias. Os valores que ele recebe em um pacote de atualização de roteamento de algum outro nó *i* contêm os custos correntes do caminho mais curto de *i* para todos os outros nós da rede. *rtupdate0()* usa esses valores recebidos para atualizar sua própria tabela de distâncias (conforme especificado pelo algoritmo de vetor de distâncias). Se seu próprio custo mínimo até outro nó mudar como resultado da atualização, o nó 0 informará essa mudança no custo mínimo a seus vizinhos diretamente conectados enviando-lhes um pacote de roteamento. Lembre-se de que no algoritmo de vetor de distâncias apenas os nós conectados diretamente trocarão pacotes de roteamento. Assim, os nós 1 e 2 vão se comunicar, mas os nós 1 e 3, não.

Rotinas semelhantes são definidas para os nós 1, 2 e 3. Assim, você escreverá oito procedimentos ao todo: *rtinit0(), rtinit1(), rtinit2(), rtinit3(), rtupdate0(), rtupdate1(), rtupdate2()* e *rtupdate3()*. Juntas, essas rotinas executarão um cálculo assíncrono, distribuído, das tabelas de distâncias para a topologia e os custos mostrados na figura apresentada anteriormente.

No site de apoio do livro você poderá encontrar todos os detalhes da tarefa de programação, bem como o código em C de que precisará para criar o ambiente simulado de hardware/software. Também está disponível uma versão da tarefa em Java.

WIRESHARK LAB

No site deste livro você encontrará duas tarefas de laboratório Wireshark, em inglês. A primeira examina a operação do protocolo IP e, em particular, o formato do datagrama IP. A segunda explora a utilização do protocolo ICMP nos comandos *ping* e Traceroute.

ENTREVISTA

Vinton G. Cerf

Vinton G. Serf é vice-presidente e evangelista-chefe da Internet para o Google. Ele trabalhou por mais 16 anos na MCI, ocupando diversos cargos, sendo o último como vice-presidente sênior de Estratégia de Tecnologia. É muito conhecido pela coautoria dos protocolos TCP/IP e da arquitetura da Internet. Durante sua gestão de 1976 a 1982 na Advanced Research Projects Agency do Departamento de Defesa dos Estados Unidos (DARPA), desempenhou um papel fundamental na liderança do desenvolvimento da Internet e de pacotes de dados e técnicas de segurança relacionadas com a Internet. Em 2005, ele recebeu a Medalha Presidencial de Liberdade dos EUA, e a Medalha Nacional de Tecnologia dos EUA em 1997. Ele é bacharel em Matemática pela Stanford University e doutor em ciência da computação pela UCLA.

O que o fez se decidir pela especialização em redes?

Eu trabalhava como programador na UCLA no final da década de 1960 com o patrocínio da Advanced Research Projects Agency do Departamento de Defesa dos Estados Unidos (na época conhecida como ARPA e hoje, como DARPA). Meu trabalho era desenvolvido no laboratório do professor Leonard Kleinrock no Network Measurement Center da recém-criada ARPAnet. O primeiro nó da ARPAnet foi instalado na UCLA em 1º de setembro de 1969. Eu era responsável pela programação de um computador utilizado para coletar informações de desempenho da ARPAnet e passá-las para comparação com modelos matemáticos e previsões de desempenho da rede.

Eu e vários outros alunos de pós-graduação éramos responsáveis pelo trabalho com o que era conhecido como protocolos de nível de hospedeiro da ARPAnet — os procedimentos e formatos que permitiriam a interação dos muitos tipos diferentes de computadores na rede. Era uma exploração fascinante de um novo mundo (para mim) de computação e comunicação distribuídas.

Quando começou a projetar o IP, você imaginava que esse protocolo tornar-se-ia tão predominante quanto é hoje?

Quando Bob Kahn e eu começamos a trabalhar nisso, em 1973, acho que estávamos muito mais preocupados com a questão central: como fazer redes de pacotes heterogêneas interagirem umas com as outras, admitindo que não poderíamos modificá-las. Esperávamos descobrir um modo que permitisse que um conjunto arbitrário de redes de comutação de pacotes fosse interligado de maneira transparente, de modo que os computadores componentes das redes pudessem se comunicar fim a fim sem precisar de nenhuma tradução entre eles. Acho que sabíamos que estávamos lidando com uma tecnologia poderosa e expansível, mas duvido que tivéssemos uma ideia muito clara do que seria o mundo com centenas de milhões de computadores todos interligados com a Internet.

Em sua opinião, qual é o futuro das redes e da Internet? Quais são os grandes obstáculos/desafios que estão no caminho do seu desenvolvimento?

Acredito que a Internet, em particular, e as redes, em geral, continuarão a proliferar. Hoje já existem evidências convincentes de que haverá bilhões de dispositivos habilitados para a Internet, entre eles equipamentos como telefones celulares, refrigeradores, PDAs, servidores residenciais, aparelhos de televisão, bem como a costumeira coleção de notebooks, servidores e assim por diante. Entre os grandes desafios estão o suporte para a mobilidade, a duração das baterias, a capacidade dos enlaces de acesso à rede e a escalabilidade ilimitada do núcleo óptico da rede. A tarefa com a qual estou

profundamente envolvido no Jet Propulsion Laboratory é o projeto de uma extensão interplanetária da Internet. Precisaremos descobrir um atalho para passar do IPv4 [endereços de 32 bits] para o IPv6 [128 bits]. A lista é comprida!

Quais pessoas o inspiraram profissionalmente?

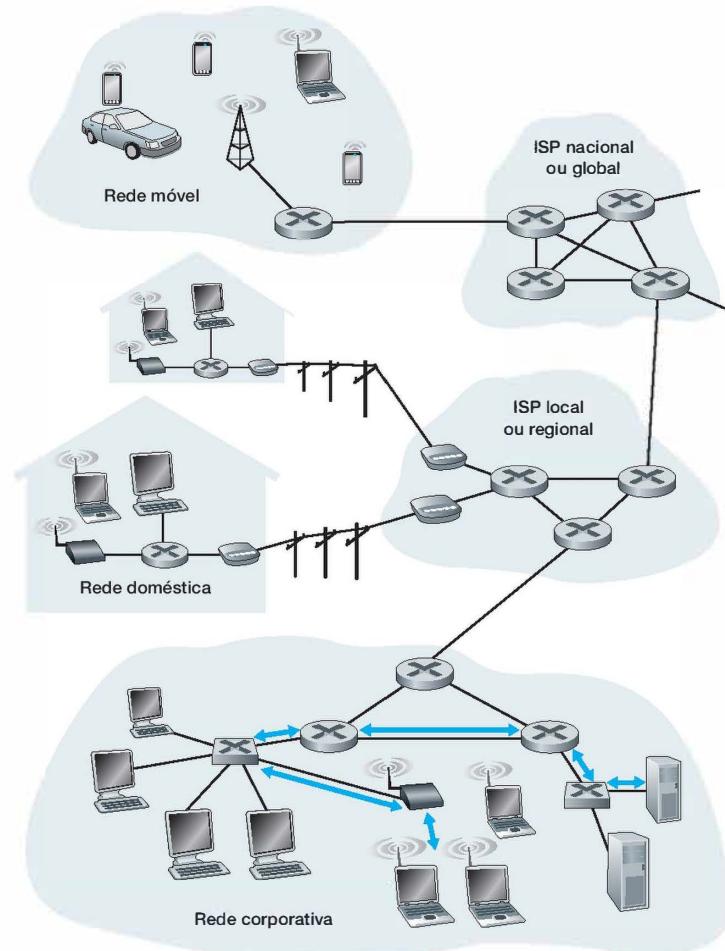
Meu colega Bob Kahn; o orientador de minha tese, Gerald Estrin; meu melhor amigo, Steve Crocker (nós nos conhecemos na escola secundária e ele me apresentou aos computadores em 1960!); e os milhares de engenheiros que continuam a evoluir a Internet ainda hoje.

Você pode dar algum conselho aos estudantes que estão ingressando no campo das redes/Internet?

Não limitem seu pensamento aos sistemas existentes — imaginem o que poderia ser possível e então comecem a trabalhar para descobrir um meio de sair do estado atual das coisas e chegar lá. Ousem sonhar; eu e alguns colegas do Jet Propulsion Laboratory estamos trabalhando no projeto de uma extensão interplanetária da Internet terrestre. A implementação dessa rede pode levar décadas, uma missão por vez, mas, usando uma paráfrase: "O homem deve tentar alcançar o que está fora do seu alcance; senão, para que existiria o céu?".

C A P Í T U L O

CAMADA DE ENLACE: **ENLACES, REDES DE** ACESSO E REDES LOCAIS

No capítulo anterior, aprendemos que a camada de rede fornece um serviço de comunicação entre dois hospedeiros *quaisquer* da rede. Entre eles, os datagramas trafegam por uma série de enlaces de comunicação, alguns com fio e alguns sem, começando no hospedeiro de origem, passando por uma série de comutadores de pacotes (comutadores e roteadores) e terminando no hospedeiro de destino. À medida que continuamos a descer a pilha de protocolos, da camada de rede até a camada de enlace, é natural que imaginemos como pacotes são enviados pelos *enlaces individuais* no caminho de comunicação fim a fim. Como os datagramas da camada de rede são encapsulados nos quadros da camada de enlace para transmissão por um único enlace? Diferentes protocolos da camada de enlace são usados em diversos enlaces no caminho de comunicação? Como os conflitos de transmissão nos enlaces de difusão podem ser resolvidos? Existe endereçamento na camada de enlace e, se houver, como o endereçamento da camada de enlace opera com o endereçamento da camada de rede, que aprendemos no Capítulo 4? E qual é exatamente a diferença entre um comutador e um roteador? Neste capítulo responderemos a essas e a outras perguntas importantes.

Ao discutir a camada de enlace, descobriremos que há dois tipos de canais completamente diferentes dessa camada. O primeiro são os canais de difusão (*broadcast*), que conectam múltiplos hospedeiros em LANs sem fio, redes por satélite e redes de acesso híbridas de cabo coaxial e de fibra (HFC). Como muitos hospedeiros são conectados ao mesmo canal de comunicação por difusão, é necessário um protocolo, denominado de acesso ao meio, para coordenar a transmissão de quadros. Em alguns casos, um controlador central pode ser usado para coordenar as transmissões; em outros, os próprios hospedeiros as coordenam. O segundo tipo é o enlace de comunicação ponto a ponto, tal como o existente entre dois roteadores conectados por um enlace de longa distância, ou entre um computador no escritório de um usuário e o comutador Ethernet próximo ao qual ele está conectado. Coordenar o acesso a um enlace ponto a ponto é mais simples; o material de referência no site deste livro possui uma discussão detalhada do Point-to-Point Protocol (PPP), que é usado em configurações que variam desde o serviço discado por uma linha telefônica até o transporte de quadros ponto a ponto de alta velocidade por enlaces de fibra ótica.

Neste capítulo estudaremos diversas tecnologias importantes da camada de enlace. Examinaremos em detalhes a detecção e correção de erros, um assunto que abordamos por alto no Capítulo 3. Consideraremos as redes de acesso múltiplo e as LANs comutadas, incluindo Ethernet, de longe a tecnologia predominante para LANs com fio. Estudaremos também as LANs virtuais e as redes de datacenter. Embora Wi-Fi e, mais frequente, as LANs sem fio sejam tópicos da camada de enlace, deixaremos nosso estudo desses assuntos importantes para o Capítulo 6.

5.1 INTRODUÇÃO À CAMADA DE ENLACE

Vamos começar com um pouco de terminologia útil. Achamos que, neste capítulo, é conveniente nos referirmos a qualquer dispositivo que rode um protocolo da camada de enlace (isto é, camada 2) como um **nó**. Os nós incluem hospedeiros, roteadores, comutadores e pontos de acesso Wi-Fi (discutidos no Capítulo 6). Também nos referiremos aos canais de comunicação que conectam nós adjacentes nos caminhos de comunicação como **enlaces**. Para levar um datagrama de um hospedeiro de origem até um de destino, o datagrama tem de ser transportado sobre cada um dos *enlaces individuais* existentes no caminho fim a fim. Como um exemplo, na rede corporativa mostrada na parte inferior da Figura 5.1, considere o envio de um datagrama de um dos hospedeiros sem fio para um dos servidores. Esse datagrama, na verdade, passará por seis enlaces: um Wi-Fi entre o hospedeiro remetente e o ponto de acesso Wi-Fi, um Ethernet entre o ponto de acesso e um comutador da camada de enlace; um entre o comutador da camada de enlace e o roteador, um entre os dois roteadores; um Ethernet entre o roteador e um comutador da camada de enlace e, por fim, um enlace Ethernet entre o comutador e o servidor.

FIGURA 5.1 SEIS SALTOS DA CAMADA DE ENLACE ENTRE HOSPEDEIRO SEM FIO E SERVIDOR

Considerando dado enlace, um nó transmissor encapsula o datagrama em um **quadro da camada de enlace** e o transmite para dentro do enlace.

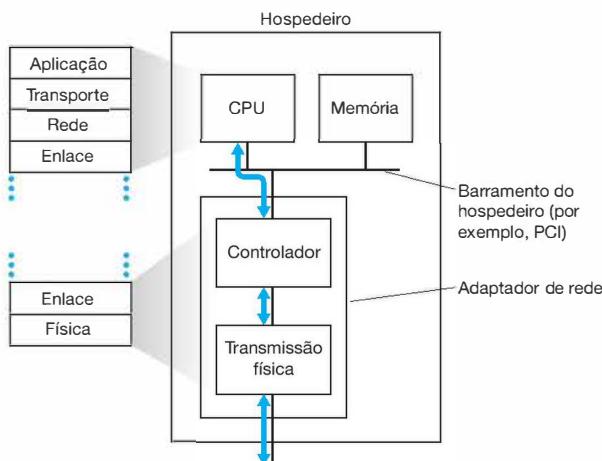
Para uma boa compreensão da camada de enlace e de como ela se relaciona com a camada de rede, vamos considerar uma analogia com um sistema de transporte. Imagine um agente de viagens que planeja uma viagem para um turista de Princeton, em Nova Jersey, até Lausanne, na Suíça. O agente decide que é mais conveniente para o turista pegar uma limusine de Princeton até o aeroporto JFK, em seguida um avião até o aeroporto de Genebra e, por fim, um trem até a estação ferroviária de Lausanne. Assim que o agente fizer as três reservas, é responsabilidade da empresa de limusines conduzir o turista de Princeton ao aeroporto JFK; é responsabilidade da companhia aérea transportar o turista do aeroporto JFK a Genebra; e é responsabilidade do trem suíço levar o turista de Genebra a Lausanne. Cada um dos três segmentos da viagem é "direto" entre duas localidades "adjacentes". Note que os três segmentos de transporte são administrados por empresas diferentes e usam meios de transporte completamente diferentes (limusine, avião e trem). Embora os meios de transporte sejam diferentes, cada um fornece o serviço básico de levar passageiros de uma localidade a outra adjacente. Nessa comparação com o transporte, o turista seria um datagrama; cada segmento de transporte, um enlace de comunicação; o meio de transporte, um protocolo da camada de enlace; e o agente de viagens, um protocolo de roteamento.

5.1.1 Os serviços fornecidos pela camada de enlace

Embora o serviço básico de qualquer camada de enlace seja mover um datagrama de um nó até um nó adjacente por um único enlace de comunicação, os detalhes do serviço podem variar de um protocolo da camada de enlace para outro. Entre os serviços que podem ser oferecidos por um protocolo da camada de enlace, estão:

- *Enquadramento de dados*. Quase todos os protocolos da camada de enlace encapsulam cada datagrama da camada de rede dentro de um quadro da camada de enlace antes de transmiti-lo pelo enlace. Um quadro consiste em um campo de dados no qual o datagrama da camada de rede é inserido, e em uma série de campos de cabeçalho. A estrutura do quadro é especificada pelo protocolo da camada de enlace. Veremos diversos formatos de quadros diferentes quando examinarmos os protocolos da camada de enlace específicos na segunda metade deste capítulo.
- *Acesso ao enlace*. Um protocolo de controle de acesso ao meio (*medium access control* — MAC) especifica as regras segundo as quais um quadro é transmitido pelo enlace. Para enlaces ponto a ponto que têm um único remetente em uma extremidade do enlace e um único receptor na outra, o protocolo MAC é simples (ou inexistente) — o remetente pode enviar um quadro sempre que o enlace estiver ocioso. O caso mais interessante é quando vários nós compartilham um único enlace de difusão — o denominado problema de acesso múltiplo. Aqui, o protocolo MAC serve para coordenar as transmissões de quadros dos muitos nós.
- *Entrega confiável*. Quando um protocolo da camada de enlace fornece serviço confiável de entrega, ele garante que vai transportar sem erro cada datagrama da camada de rede pelo enlace. Lembre-se de que certos protocolos da camada de transporte (como o TCP) também fornecem um serviço confiável de entrega. Semelhante ao que acontece com um serviço confiável de entrega da camada de transporte, consegue-se um serviço confiável de entrega da camada de enlace com reconhecimentos e retransmissões (veja a Seção 3.4). Um serviço confiável de entrega da camada de enlace é muito usado por enlaces que costumam ter altas taxas de erros, como é o caso de um enlace sem fio, com a finalidade de corrigir um erro localmente, no enlace no qual o erro ocorre, em vez de forçar uma retransmissão fim a fim dos dados por um protocolo da camada de transporte ou de aplicação. Contudo, a entrega confiável da camada de enlace pode ser considerada uma sobrecarga desnecessária para enlaces de baixa taxa de erros, incluindo enlaces de fibra, enlaces coaxiais e muitos enlaces de pares de fios trançados de cobre. Por essa razão, muitos protocolos da camada de enlace com fio não fornecem um serviço de entrega confiável.
- *Detecção e correção de erros*. O hardware da camada de enlace de um nó receptor pode decidir incorretamente que um bit de um quadro é zero quando foi transmitido como 1 e vice-versa. Esses erros de bits

são introduzidos por atenuação de sinal e ruído eletromagnético. Como não há necessidade de repassar um datagrama que tem um erro, muitos protocolos da camada de enlace oferecem um mecanismo para detectar a presença de tais erros. Isso é feito obrigando o nó transmissor a enviar bits de detecção de erros no quadro e o nó receptor a realizar uma verificação de erros. Lembre-se de que nos Capítulos 3 e 4 dissemos que as camadas de transporte e de rede da Internet também fornecem um serviço limitado de detecção de erros — a soma de verificação da Internet. A detecção de erros na camada de enlace geralmente é mais sofisticada e é executada em hardware. A correção de erros é semelhante à detecção de erros, exceto que um receptor não só detecta quando ocorreram os erros no quadro, mas também determina exatamente em que lugar do quadro ocorreram (e, então, os corrige).

5.1.2 Onde a camada de enlace é implementada?

Antes de mergulharmos em nosso detalhado estudo sobre a camada de enlace, vamos considerar a questão de onde esta é implementada. Daqui em diante focaremos em um sistema final, visto que já aprendemos no Capítulo 4 como a camada de enlace é implementada em uma placa de linha de um roteador. A camada de enlace de um computador deve ser implementada no hardware ou no software? Deve ser implementada em uma placa ou chip separado e como ocorre a interface com o resto do hardware de um hospedeiro e com os componentes de sistemas operacionais?

A Figura 5.2 mostra a arquitetura típica de um hospedeiro. Na maior parte, a camada de enlace é implementada em um **adaptador de rede**, às vezes também conhecido como **placa de interface de rede (NIC)**. No núcleo do adaptador de rede está o controlador da camada de enlace, em geral um único chip de sistema especial, que executa vários serviços da camada de enlace (enquadramento, acesso ao enlace, detecção de erros etc.). Dessa forma, muito da funcionalidade do controlador da camada de enlace é realizado em hardware. Por exemplo, o controlador da Intel 8254*x* [Intel, 2012] implementa os protocolos Ethernet, os quais estudaremos na Seção 5.5; o controlador Atheros AR5006 [Atheros, 2012] implementa os protocolos Wi-Fi 802.11 que estudaremos no Capítulo 6. Até o final dos anos 1990, a maioria dos adaptadores de rede eram placas fisicamente separadas (como a placa PCMCIA ou uma placa *plug-in* que se encaixa em um compartimento para cartão PCI de um computador), porém agora cada vez mais adaptadores de rede estão sendo integrados à placa-mãe do hospedeiro — uma configuração chamada LAN-na-placa-mãe.

No lado transmissor, o controlador separa um datagrama que foi criado e o armazena na memória do hospedeiro por camadas mais altas da pilha de protocolos, encapsula o datagrama em um quadro da camada de enlace (preenchendo os vários campos do quadro), e então transmite o quadro para um enlace de comunicação, seguindo o protocolo de acesso ao enlace. No lado receptor, um controlador recebe todo o quadro e extrai o datagrama da camada de rede. Se a camada de enlace efetuar uma verificação de erros, é o controlador transmissor que estabelece os bits de detecção de erros no cabeçalho de quadro e é o controlador receptor que executa a verificação de erros.

A Figura 5.2 mostra um adaptador de rede conectado ao barramento do computador (por exemplo, barramento PCI ou PCI-X), que se parece muito com qualquer outro dispositivo de entrada/saída dos outros componentes do hospedeiro. A Figura 5.2 mostra também que, enquanto a maior parte da camada de enlace é executada em hardware, parte dela é implementada em software que é executada na CPU do hospedeiro. Os componentes do software da camada de enlace implementam uma funcionalidade da camada de enlace de um nível mais alto, montando informações de endereçamento da camada de enlace e ativando o hardware do controlador. No lado receptor, o software da camada de enlace responde a interrupções do controlador (por exemplo, pelo recebimento de um ou mais quadros), lida com condições de erro e passa o datagrama para a camada de rede, mais acima. Assim, a camada de enlace é uma combinação de hardware e software — o lugar na pilha de protocolos, onde o software encontra o hardware. Intel [2012] fornece um panorama legível (assim como descrições detalhadas) do controlador 8254x do ponto de vista de uma programação de software.

FIGURA 5.2 ADAPTADOR DE REDE: SEU RELACIONAMENTO COM O RESTO DOS COMPONENTES DO HOSPEDEIRO E A FUNCIONALIDADE DA PILHA DE PROTOCOLOS

5.2 TÉCNICAS DE DETECÇÃO E CORREÇÃO DE ERROS

Na seção anterior, observamos que **detecção e correção de erros no nível de bits** — detecção e correção da alteração de bits em um quadro da camada de enlace enviado de um nó para outro nó vizinho fisicamente ligado a ele — são dois serviços com fornecidos frequência pela camada de enlace. Vimos no Capítulo 3 que serviços de detecção e correção de erros também são frequentemente oferecidos na camada de transporte. Nesta seção, examinaremos algumas das técnicas mais simples que podem ser usadas para detectar e, em alguns casos, corrigir esses erros de bits. Como a teoria e a implementação dessas técnicas é um assunto tratado detalhadamente em muitos livros (como Schwartz [1980] ou Bertsekas [1991]), nossa abordagem terá de ser breve. Nossa meta é desenvolver uma visão intuitiva das capacidades que as técnicas de detecção e correção de erros fornecem e ver como algumas técnicas simples funcionam e são usadas na prática na camada de enlace.

A Figura 5.3 ilustra o cenário de nosso estudo. No nó remetente, para que os dados, *D*, fiquem protegidos contra erros de bits, eles são aumentados com bits de detecção e de correção (*error detection-and-correction — EDC*). Em geral, os dados que devem ser protegidos incluem não somente o datagrama passado para baixo a partir da camada de rede para transmissão pelo enlace, mas também informações de endereçamento da camada de enlace, números de sequência e outros campos do cabeçalho do quadro de enlace. Tanto *D* como *EDC* são enviados ao nó receptor em um quadro no nível de enlace. No nó receptor, são recebidas sequências de bits, *D'* e *EDC'*. Note que *D'* e *EDC'* podem ser diferentes dos *D* e *EDC* originais, como resultado de inversões nos bits em trânsito.

O desafio do receptor é determinar se *D'* é ou não igual ao *D* original, uma vez que recebeu apenas *D'* e *EDC'*. A exata sintaxe da decisão do receptor na Figura 5.4 (perguntamos se um erro foi detectado, e não se um erro foi cometido!) é importante. Técnicas de detecção e correção permitem que o receptor descubra a ocorrência de erros de bits às vezes, *mas não sempre*. Mesmo com a utilização de bits de detecção de erros, ainda pode haver **erros de bits não detectados**, isto é, o receptor pode não perceber que a informação recebida contém erros de bits. Por conseguinte, o receptor poderá entregar um datagrama corrompido à camada de rede ou não perceber que o conteúdo de um campo no cabeçalho do quadro foi corrompido. Assim, é preciso escolher um esquema de detecção de erros para o qual a probabilidade dessas ocorrências seja pequena. Em geral, técnicas mais sofisticadas de detecção e correção de erros (isto é, as que têm uma probabilidade menor de permitir erros de bits não detectados) ficam sujeitas a uma sobrecarga maior — é preciso mais processamento para calcular e transmitir um número maior de bits de detecção e correção de erros.

FIGURA 5.3 CENÁRIO DE DETECÇÃO E CORREÇÃO DE ERROS

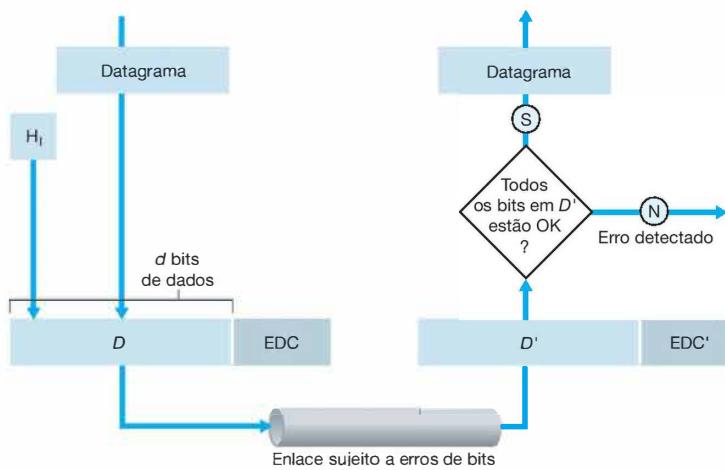
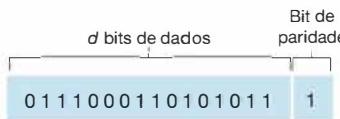



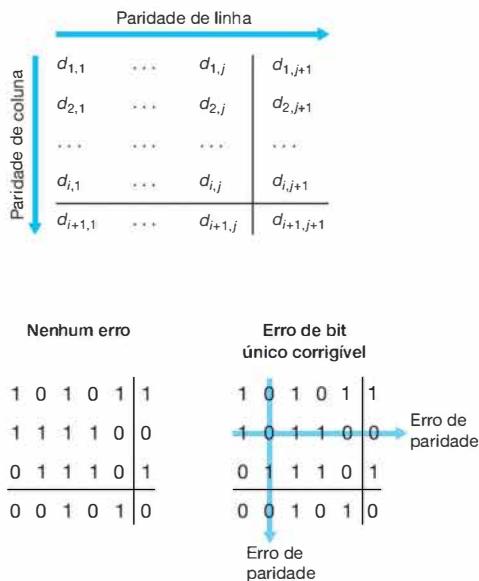
FIGURA 5.4 PARIDADE PAR USANDO UM BIT

Vamos examinar agora três técnicas de detecção de erros nos dados transmitidos — verificações de paridade (para ilustrar as ideias básicas da detecção e correção de erros), métodos de soma de verificação (que são mais empregados na camada de transporte) e verificações de redundância cíclica (CRCs) (que são em geral empregadas na camada de enlace, nos adaptadores).

5.2.1 Verificações de paridade

Talvez a maneira mais simples de detectar erros seja utilizar um único **bit de paridade**. Suponha que a informação a ser enviada, *D* na Figura 5.4, tenha *d* bits. Em um esquema de paridade par, o remetente apenas inclui um bit adicional e escolhe o valor desse bit de modo que o número total de "1" nos $d + 1$ bits (a informação original mais um bit de paridade) seja par. Em esquemas de paridade ímpar, o valor do bit de paridade é escolhido para que haja um número ímpar de "1". A Figura 5.4 ilustra um esquema de paridade par com o bit de paridade armazenado em um campo separado.

Com um único bit de paridade, a operação do receptor também é simples. O receptor precisa apenas contar quantos "1" há nos $d + 1$ bits recebidos. Se, utilizando um esquema de paridade par, for encontrado um número ímpar de bits de valor 1, o receptor saberá que ocorreu pelo menos um erro de bit. Mais precisamente, ele saberá que ocorreu algum número *ímpar* de erros de bit.

Mas o que acontecerá se ocorrer um número par de erros de bit? É bom que você se convença de que isso resultaria em um erro não detectado. Se a probabilidade de erro de bits for pequena e se for razoável admitir que os erros ocorrem independentemente entre um bit e o bit seguinte, a probabilidade de haver vários erros de bits em um pacote seria bastante pequena. Nesse caso, um único bit de paridade poderia ser suficiente. Contudo, medições demonstraram que, em vez de acontecerem independentemente, os erros em geral se aglomeram em

"rajadas". Na condição de rajada de erros, a probabilidade de haver erros não detectados em um quadro protegido por um esquema de paridade de bit único pode chegar perto de 50% [Spragins, 1991]. Claro que é necessário um esquema de detecção de erros mais robusto (e, felizmente, é o que se usa na prática!). Mas, antes de examinarmos os esquemas usados na prática, vamos considerar uma generalização simples da paridade de bit único que nos dará uma ideia das técnicas de correção.

A Figura 5.5 mostra uma generalização bidimensional do esquema de paridade de bit único. Nessa figura, os d bits de D são divididos em i linhas e j colunas. Um valor de paridade é calculado para cada linha e para cada coluna. Os $i + j + 1$ bits de paridade resultantes compreendem os bits de detecção de erros do quadro da camada de enlace.

Suponha agora que ocorra um erro de bit único nos d bits originais de informação. Com esse esquema de **paridade bidimensional**, tanto a paridade da coluna quanto a da linha que contiver o bit modificado estarão com erro. O receptor então não só pode *detectar* que ocorreu um erro de um bit único, mas também usar os índices da linha e da coluna com erros de paridade para realmente identificar o bit que foi corrompido e *corrigir* aquele erro! A Figura 5.5 mostra um exemplo no qual o bit com valor 1 na posição (2, 2) está corrompido e mudou para um 0 — um erro que não somente é detectável, como também corrigível no receptor. Embora nossa discussão tenha focalizado os d bits originais de informação, um erro único nos próprios bits de paridade também é detectável e corrigível. A paridade bidimensional também pode detectar (mas não corrigir!) qualquer combinação de dois erros em um pacote. Outras propriedades do esquema de paridade bidimensional são discutidas nos exercícios ao final deste capítulo.

A capacidade do receptor para detectar e corrigir erros é conhecida como **correção de erros antecipada** (*forward error correction* — FEC). Essas técnicas são usadas na armazenagem de áudio e em equipamentos de reprodução, como CDs de áudio. Em um ambiente de rede, as técnicas FEC podem ser usadas isoladamente ou em conjunto com as técnicas ARQ, que examinamos no Capítulo 3. As técnicas FEC são valiosas porque podem reduzir o número exigido de retransmissões do remetente. Talvez o mais importante seja que elas permitem imediata correção de erros no receptor. Isso evita ter de esperar pelo atraso de propagação da viagem de ida e volta de que o remetente precisa para receber um pacote NAK e para que um pacote retransmitido se propague de volta

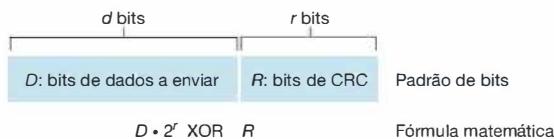
FIGURA 5.5 PARIDADE PAR BIDIMENSIONAL

Paridade de linha

Paridade de coluna

$d_{1,1}$	...	$d_{1,j}$	$d_{1,j+1}$
$d_{2,1}$	...	$d_{2,j}$	$d_{2,j+1}$
...	...	...	...
$d_{i,1}$	...	$d_{i,j}$	$d_{i,j+1}$
$d_{i+1,1}$	...	$d_{i+1,j}$	$d_{i+1,j+1}$

ao receptor — uma vantagem potencialmente importante para aplicações de rede em tempo real [Rubenstein, 1998] ou enlaces (como enlaces no espaço sideral) com longos atrasos de propagação. Entre as pesquisas que examinaram a utilização da FEC em protocolos de controle de erros estão as de Biersack [1992]; Nonnenmacher [1998]; Byers [1998]; Shacham [1990].

5.2.2 Métodos de soma de verificação

Em técnicas de soma de verificação, os *d* bits de dados na Figura 5.4 são tratados como uma sequência de números inteiros de *k* bits. Um método simples de soma de verificação é somar esses inteiros de *k* bits e usar o total resultante como bits de detecção de erros. A **soma de verificação da Internet** é baseada nessa técnica — bytes de dados são tratados como inteiros de 16 bits e somados. O complemento de 1 dessa soma forma, então, a soma de verificação da Internet, que é carregada no cabeçalho do segmento. Como discutido na Seção 3.3, o receptor verifica a soma de verificação calculando os complementos de 1 da soma dos dados recebidos (inclusive a soma de verificação) e averiguando se o resultado contém somente bits 1. Se qualquer um dos bits for 0, isso indicará um erro. O RFC 1071 discute em detalhes o algoritmo da soma de verificação da Internet e sua implementação. Nos protocolos TCP e UDP, a soma de verificação da Internet é calculada sobre todos os campos (incluindo os de cabeçalho e de dados). No IP, a soma de verificação é calculada sobre o cabeçalho IP (já que o segmento UDP ou TCP tem sua própria soma de verificação). Em outros protocolos, o XTP, por exemplo Strayer [1992], uma soma de verificação é calculada sobre o cabeçalho e outra soma de verificação é calculada sobre o pacote inteiro.

Métodos de soma de verificação exigem relativamente pouca sobrecarga no pacote. Por exemplo, as de TCP e UDP utilizam apenas 16 bits. Contudo, oferecem proteção um tanto baixa contra erros comparados com a verificação de redundância cíclica, discutida mais adiante, que é utilizada com frequência na camada de enlace. Uma pergunta que surge naturalmente neste ponto é: por que a soma de verificação é utilizada na camada de transporte e a verificação de redundância cíclica é utilizada na camada de enlace? Lembre-se de que a camada de transporte em geral é executada em software, como parte do sistema operacional de um hospedeiro. Como a detecção de erros na camada de transporte é realizada em software, é importante que o esquema de detecção de erros seja simples e rápido como a soma de verificação. Por outro lado, a detecção de erro na camada de enlace é implementada em hardware dedicado nos adaptadores, que podem rodar muito rápido as mais complexas operações de CRC. Feldmeier [1995] apresenta técnicas de implementação rápida em software não só para códigos de soma de verificação ponderada, mas também para CRC (veja a seguir) e outros códigos.

5.2.3 Verificação de redundância cíclica (CRC)

Uma técnica de detecção de erros muito usada nas redes de computadores de hoje é baseada em **códigos de verificação de redundância cíclica** (*cyclic redundancy check* — CRC). Códigos de CRC também são conhecidos como **códigos polinomiais**, já que é possível considerar a cadeia de bits a ser enviada como um polinômio cujos coeficientes são os valores 0 e 1 na cadeia, sendo as operações interpretadas como aritmética polinomial.

Códigos de CRC funcionam da seguinte forma. Considere a parcela de *d* bits de dados, *D*, que o nó remetente quer enviar para o nó receptor. O remetente e o receptor devem, primeiro, concordar com um padrão de $r + 1$ bits, conhecido como um **gerador**, que denominaremos *G*. Vamos exigir que o bit mais significativo (o da extrema esquerda) de *G* seja um 1. A ideia fundamental por trás dos códigos de CRC é mostrada na Figura 5.6. Para dada parcela de dados, *D*, o remetente escolherá *r* bits adicionais, *R*, e os anexará a *D* de modo que o padrão de $d + r$ bits resultante (interpretado como um número binário) seja divisível exatamente por *G* (por exemplo, sem nenhum remanescente), usando aritmética de módulo 2. Assim, o processo de verificação de erros com CRC é simples: o receptor divide os $d + r$ bits recebidos por *G*. Se o resto for diferente de zero, o receptor saberá que ocorreu um erro; caso contrário, os dados são aceitos como corretos.

FIGURA 5.6 CÓDIGOS DE CRC

Todos os cálculos de CRC são feitos por aritmética de módulo 2 sem "vai 1" nas adições nem "empresta 1" nas subtrações. Isso significa que a adição e a subtração são idênticas e ambas são equivalentes à operação *ou exclusivo* (XOR) bit a bit dos operandos. Por exemplo,

```
1011 XOR 0101 = 1110
1001 XOR 1101 = 0100
```

Também de modo semelhante temos:

```
1011 – 0101 = 1110
1001 – 1101 = 0100
```

A multiplicação e a divisão são as mesmas da base 2, exceto que, em qualquer adição ou subtração exigida, não se emprestam nem se tomam emprestadas casas. Como na aritmética binária comum, a multiplicação por 2^k desloca um padrão de bits para a esquerda por k casas. Assim, dados D e R, a quantidade $D \cdot 2^r$ XOR R dá como resultado o padrão de bits $d + r$ mostrado na Figura 5.6. Usaremos a representação algébrica do padrão de bits $d + r$ da Figura 5.6 em nossa discussão a seguir.

Vamos agora voltar à questão crucial de como o remetente calcula R. Lembre-se de que queremos descobrir um R tal que exista um n tal que

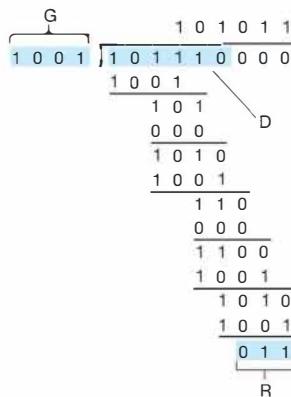
$$D \cdot 2^r \text{ XOR } R = nG$$

Isto é, devemos escolher um R tal que G divida $D \cdot 2^r$ XOR sem resto. Se usarmos a operação XOR (isto é, adicionarmos com módulo 2, sem vai 1) de R com ambos os lados da equação anterior, teremos:

$$D \cdot 2r = nG \text{ XOR } R$$

Essa equação nos diz que, se dividirmos $D \cdot 2^r$ por G, o valor do resto será exatamente R. Em outras palavras, podemos calcular R como

$$R = \text{resto } \frac{D \cdot 2^r}{G}$$

A Figura 5.7 ilustra esses cálculos para o caso de $D = 101110$, $d = 6$, $G = 1001$ e $r = 3$. Os 9 bits transmitidos nesse caso são 101110 011. Você deve fazer esses cálculos e também verificar se, na verdade, $D \cdot 2^r = 101011 \cdot G$ XOR R.

Padrões internacionais foram definidos para geradores G de 8, 12, 16 e 32 bits. O padrão CRC-32 de 32 bits, que foi adotado em uma série de protocolos do IEEE da camada de enlace, usa um gerador igual a

$$G_{\text{CRC-32}} = 100000100110000010001110110110111$$

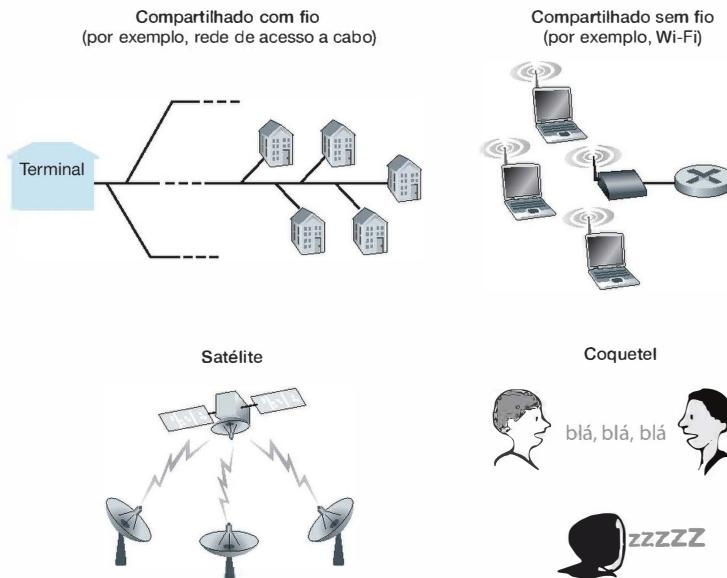
Cada padrão de CRC pode detectar erros de rajada de menos do que $r + 1$ bits. (Isso significa que todos os erros de bits consecutivos de r bits ou menos serão detectados.) Além disso, em hipóteses apropriadas, uma rajada de comprimento maior do que $r + 1$ bits é detectada com probabilidade de $1 - 0{,}5^r$. Cada um dos padrões de CRC também pode detectar qualquer número ímpar de erros de bits. Veja em Williams [1993] uma discussão

sobre a realização de verificações de CRC. A teoria por trás dos códigos de CRC e de códigos até mais poderosos ultrapassa o escopo deste livro. O livro de Schwartz [1980] oferece uma excelente introdução a esse tópico.

FIGURA 5.7 UM EXEMPLO DE CÁLCULO DE CRC

5.3 ENLACES E PROTOCOLOS DE ACESSO MÚLTIPLO

Na introdução deste capítulo, observamos que há dois tipos de enlaces de redes: ponto a ponto e enlaces de difusão. Um **enlace ponto a ponto** consiste em um único remetente em uma extremidade do enlace e um único receptor na outra. Muitos protocolos da camada de enlace foram projetados para enlaces ponto a ponto; o PPP (protocolo ponto a ponto) e um controle de ligação de dados de alto nível (HDCL) são alguns que examinaremos mais adiante neste capítulo. O segundo tipo, o **enlace de difusão**, pode ter vários nós remetentes e receptores, todos conectados ao mesmo canal de transmissão único e compartilhado. O termo difusão é usado aqui porque, quando qualquer um dos nós transmite um quadro, o canal propaga o quadro por difusão e cada nó recebe uma cópia. A Ethernet e as LANs sem fio são exemplos de tecnologias de difusão da camada de enlace. Nesta seção, vamos nos afastar um pouco dos protocolos específicos dessa camada e examinar, primeiro, um problema de importância fundamental para a camada de enlace de dados: como coordenar o acesso de vários nós remetentes e receptores a um canal de difusão compartilhado — o **problema do acesso múltiplo**. Canais de difusão são muito usados em LANs, redes que estão geograficamente concentradas em um único prédio (ou empresa, ou *campus*). Assim, no final da seção, examinaremos também como os canais de acesso múltiplo são usados em LANs.

Todos estamos familiarizados com a noção de difusão — a televisão usa essa tecnologia desde que foi inventada. Mas a televisão tradicional é uma difusão unidirecional (isto é, um nó fixo que transmite para muitos nós receptores), ao passo que os nós em um canal de difusão de uma rede de computadores tanto podem enviar quanto receber. Talvez uma analogia humana mais apropriada para um canal de difusão seja um coquetel, no qual muitas pessoas se encontram em uma sala grande para falar e ouvir (e o ar fornece o meio de transmissão). Outra boa analogia é algo com que muitos leitores estão familiarizados — uma sala de aula, onde professor(es) e estudante(s) compartilham, de maneira semelhante, o mesmo e único meio de transmissão por difusão. Um problema fundamental em ambos os cenários é determinar quem fala (isto é, quem transmite pelo canal) e quando fala. Como seres humanos, desenvolvemos uma série elaborada de protocolos para compartilhar o canal de difusão:

"Dê a todos uma oportunidade de falar."
"Não fale até que alguém fale com você."
"Não monopolize a conversa."
"Levante a mão se tiver uma pergunta a fazer."
"Não interrompa uma pessoa quando ela estiver falando."
"Não durma quando alguém estiver falando."

Redes de computadores têm protocolos semelhantes — denominados **protocolos de acesso múltiplo** —, pelos quais os nós regulam sua transmissão pelos canais de difusão compartilhados. Como ilustra a Figura 5.8, os protocolos de acesso múltiplo são necessários em diversos cenários de rede, que inclui redes de acesso com fio e sem fio, além de redes por satélite. Embora tecnicamente cada nó acesse o canal de difusão por meio de seu adaptador, nesta seção vamos nos referir ao nó como o dispositivo de envio e de recepção. Na prática, centenas ou até milhares de nós podem se comunicar diretamente por um canal de difusão.

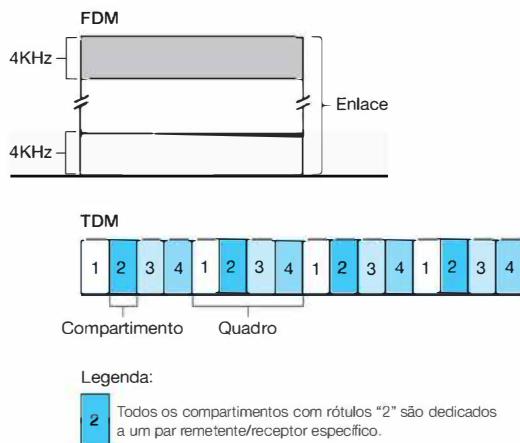
Como todos os nós têm a capacidade de transmitir quadros, mais do que dois podem transmitir quadros ao mesmo tempo. Quando isso acontece, todos os nós recebem vários quadros ao mesmo tempo, isto é, os quadros transmitidos **colidem** em todos os receptores. Em geral, quando há uma colisão, nenhum dos nós receptores consegue perceber algum sentido nos quadros que foram transmitidos; de certo modo, os sinais dos quadros que colidem ficam inextricavelmente embaralhados. Assim, todos os quadros envolvidos na colisão são perdidos e o canal de difusão é desperdiçado durante o intervalo de colisão. É claro que, se muitos nós querem transmitir quadros com frequência, muitas transmissões resultarão em colisões e grande parte da largura de banda do canal de difusão será desperdiçada.

Para assegurar que o canal de difusão realize trabalho útil quando há vários nós ativos, é preciso coordenar, de algum modo, as transmissões desses nós ativos. Essa tarefa de coordenação é de responsabilidade do protocolo de acesso múltiplo. Durante os últimos 40 anos, milhares de artigos e centenas de teses foram escritos sobre tais protocolos; um levantamento abrangente dos primeiros 20 anos desse volume de trabalho pode ser encontrado

FIGURA 5.8 VÁRIOS CANAIS DE ACESSO MÚLTIPLO

em Rom [1990]. Além disso, a pesquisa ativa sobre protocolos de acesso múltiplo continua por causa do surgimento contínuo de novos tipos de enlaces, em particular novos enlaces sem fio.

Durante anos, dezenas de protocolos de acesso múltiplo foram executados em diversas tecnologias da camada de enlace. Não obstante, podemos classificar praticamente qualquer protocolo de acesso múltiplo em uma das seguintes categorias: **protocolos de divisão de canal**, **protocolos de acesso aleatório** e **protocolos de revezamento**. Examinaremos essas categorias nas três subseções seguintes.

Vamos concluir essa visão geral mencionando que, idealmente, um protocolo de acesso múltiplo para um canal de difusão com velocidade de R bits por segundo tem as seguintes características desejáveis:

1. Quando apenas um nó tem dados para enviar, esse nó tem uma vazão de R bit/s.
2. Quando M nós têm dados para enviar, cada um desses nós tem uma vazão de R/M bits/s. Isso não significa necessariamente que cada um dos M nós sempre terá uma velocidade instantânea de R/M, mas que cada nó deverá ter uma velocidade média de transmissão de R/M durante algum intervalo de tempo adequadamente definido.
3. O protocolo é descentralizado, isto é, não há um nó mestre que represente um único ponto de falha para a rede.
4. O protocolo é simples para que sua implementação seja barata.

5.3.1 Protocolos de divisão de canal

Lembre-se de que na Seção 1.3 dissemos que a multiplexação por divisão de tempo (TDM) e a multiplexação por divisão de frequência (FDM) são duas técnicas que podem ser usadas para dividir a largura de banda de um canal de difusão entre todos os nós que compartilham esse canal. Como exemplo, suponha que o canal suporte N nós e que a velocidade de transmissão do canal seja R bits/s. O protocolo TDM divide o tempo em **quadros temporais**, os quais depois divide em N **compartimentos de tempo**. (O quadro temporal TDM não deve ser confundido com a unidade de dados da camada de enlace trocada entre adaptadores remetentes e receptores, que também é denominada um quadro. Para diminuir a confusão, nesta seção vamos nos referir à unidade de dados trocada na camada de enlace como um pacote.) Cada compartimento de tempo é, então, atribuído a um dos N nós. Sempre que um nó tiver um pacote para enviar, ele transmite os bits do pacote durante o compartimento atribuído a ele no quadro rotativo TDM. Normalmente, os tamanhos dos quadros são escolhidos de modo que um único quadro possa ser transmitido durante um compartimento de tempo. A Figura 5.9 mostra um exemplo de TDM simples de quatro nós. Voltando à analogia do coquetel, um coquetel regulamentado por TDM permitiria que um dos convidados falasse durante um período de tempo fixo; em seguida, permitiria que outro convidado falasse pelo mesmo período de tempo e assim por diante. Quando todos tivessem tido sua chance de falar, o padrão seria repetido.

O protocolo TDM é atraente, pois elimina colisões e é perfeitamente justo: cada nó ganha uma velocidade de transmissão dedicada de R/N bits/s durante cada quadro temporal. Contudo, ele tem duas desvantagens importantes. A primeira é que um nó fica limitado a uma velocidade média de R/N bits/s, mesmo quando ele é o único nó com pacotes para enviar. A segunda é que o nó deve sempre esperar sua vez na sequência de transmissão — de novo, mesmo quando ele é o único com um quadro a enviar. Imagine um convidado que é o único que tem algo a dizer (e imagine uma situação ainda mais rara, em que todos na festa querem ouvir o que aquela pessoa tem a dizer). É óbvio que o protocolo TDM seria uma má escolha para um protocolo de acesso múltiplo para essa festa em particular.

Enquanto o protocolo TDM compartilha o canal de difusão no tempo, o protocolo FDM divide o canal de R bits/s em frequências diferentes (cada uma com uma largura de banda de R/N) e reserva cada frequência a um dos N nós, criando, desse modo, N canais menores de R/N bits/s a partir de um único canal maior de R bits/s. O protocolo FDM compartilha as vantagens do protocolo TDM — evita colisões e divide a largura de banda com justiça entre os N nós. Porém, também compartilha uma desvantagem principal com o protocolo TDM — um nó é limitado a uma largura de banda R/N, mesmo quando é o único nó que tem pacotes a enviar.

FIGURA 5.9 UM EXEMPLO DE TDM E FDM DE QUATRO NÓS

Um terceiro protocolo de divisão de canal é o protocolo de **acesso múltiplo por divisão de código** (*code division multiple access* — CDMA). Enquanto os protocolos TDM e FDM atribuem aos nós intervalos de tempo e frequências, respectivamente, CDMA atribui um *código* diferente a cada nó. Então, cada nó usa seu código exclusivo para codificar os bits de dados que envia. Se os códigos forem escolhidos com cuidado, as redes CDMA terão a maravilhosa propriedade de permitir que nós diferentes transmitam *simultaneamente* e, ainda assim, consigam que seus receptores respectivos recebam corretamente os bits codificados pelo remetente (supondo que o receptor conheça o código do remetente), a despeito das interferências causadas pelas transmissões dos outros nós. O CDMA vem sendo usado em sistemas militares há algum tempo (por suas propriedades anti-interferências) e agora está bastante difundido para uso civil. Como a utilização do CDMA está muito ligada a canais sem fio, adiaremos a discussão de seus detalhes técnicos para o Capítulo 6. Por enquanto, basta saber que códigos CDMA, assim como compartimentos de tempo em TDM e frequências em FDM, podem ser alocados a usuários de canais de múltiplo acesso.

5.3.2 Protocolos de acesso aleatório

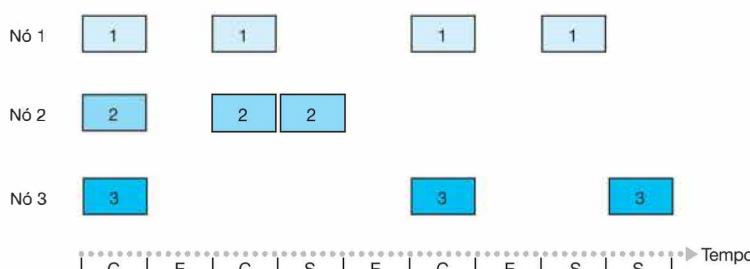
A segunda classe geral de protocolos de acesso múltiplo são os protocolos de acesso aleatório. Com um protocolo de acesso aleatório, um nó transmissor sempre transmite à taxa total do canal, isto é, R bits/s. Quando há uma colisão, cada nó envolvido nela retransmite repetidamente seu quadro (isto é, pacote) até que este passe sem colisão. Mas, quando um nó sofre uma colisão, ele nem sempre retransmite o quadro de imediato. *Em vez disso, ele espera um tempo aleatório antes de retransmitir o quadro.* Cada nó envolvido em uma colisão escolhe atrasos aleatórios independentes. Como após uma colisão os tempos de atraso são escolhidos de modo independente, é possível que um dos nós escolha um atraso mais curto o suficiente do que os atrasos dos outros nós em colisão e, portanto, consiga passar seu quadro discretamente para dentro do canal, sem colisão.

Há dezenas, se não centenas, de protocolos de acesso aleatório descritos na literatura [Rom, 1990; Bertsekas, 1991]. Nesta seção, descreveremos alguns dos mais usados — os protocolos ALOHA [Abramson, 1970; 1985] e os de acesso múltiplo com detecção de portadora (CSMA) [Kleinrock, 1975b]. Ethernet [Metcalfe, 1976] é uma variante popular e muito disseminada do protocolo CSMA.

Slotted ALOHA

Vamos começar nosso estudo de protocolos de acesso aleatório com um dos mais simples deles, o *slotted* ALOHA. Em nossa descrição, admitiremos o seguinte:

- Todos os quadros consistem em exatamente L bits.
- O tempo é dividido em intervalos (*slots*) de tamanho L/R segundos (isto é, um intervalo é igual ao tempo de transmissão de um quadro).
- Os nós começam a transmitir quadros somente no início dos intervalos.
- Os nós são sincronizados de modo que cada nó sabe onde os intervalos começam.
- Se dois ou mais nós colidirem em um intervalo, então todos os nós detectarão o evento de colisão antes do término do intervalo.

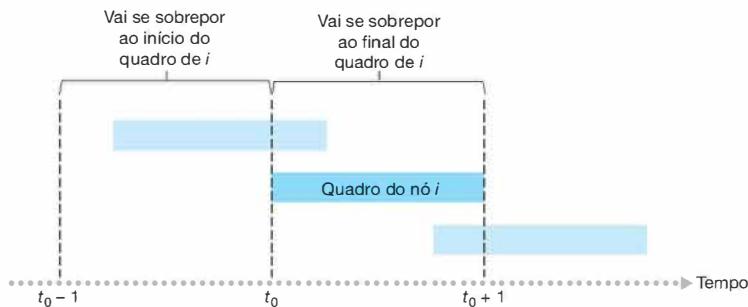
Seja p uma probabilidade, isto é, um número entre 0 e 1. O funcionamento do *slotted* ALOHA em cada nó é simples:

- Quando o nó tem um novo quadro para enviar, espera até o início do próximo intervalo e transmite o quadro inteiro no intervalo.
- Se não houver colisão, o nó terá transmitido seu quadro com sucesso e, assim, não precisará considerar a retransmissão. (Ele pode preparar um novo quadro para transmitir, se tiver algum.)
- Se houver uma colisão, o nó a detectará antes do final do intervalo. Ele retransmitirá seu quadro em cada intervalo subsequente com probabilidade p até que o quadro seja transmitido sem colisão.

Por retransmissão com probabilidade p, queremos dizer que o nó de fato joga uma moeda viciada; coroa corresponde a "retransmitir", o que ocorre com probabilidade p, enquanto cara corresponde a "pule o intervalo e jogue a moeda novamente no próximo intervalo", o que ocorre com probabilidade $(1 - p)$. Todos os nós envolvidos na colisão jogam suas moedas independentemente.

Na aparência o *slotted* ALOHA teria muitas vantagens. Ao contrário da divisão de canal, esse protocolo permite que um único nó transmita continuamente à taxa total do canal, R, quando ele for o único nó ativo. (Diz-se que um nó é ativo quanto tem quadros a enviar.) O *slotted* ALOHA também é altamente descentralizado, porque cada nó detecta colisões e decide de modo independente quando retransmitir. (No entanto, requer que os intervalos sejam sincronizados nos nós; em breve discutiremos uma versão sem intervalos do protocolo ALOHA [*unslotted* ALOHA], bem como protocolos CSMA — nenhum deles requer essa sincronização e, portanto, são totalmente descentralizados.) O *slotted* ALOHA é também um protocolo extremamente simples.

O *slotted* ALOHA funciona bem quando há apenas um nó ativo, mas qual é sua eficiência quando há vários? Nesse caso, há duas preocupações possíveis quanto à eficiência. A primeira, como mostra a Figura 5.10, é que, quando há vários nós ativos, certa fração dos intervalos terá colisões e, portanto, será "desperdiçada". A segunda é que outra fração dos intervalos estará *vazia* porque todos os nós ativos evitarão transmitir como resultado da po-

FIGURA 5.10 NÓS 1, 2 E 3 COLIDEM NO PRIMEIRO INTERVALO. O NÓ 2 FINALMENTE É BEM-SUCEDIDO NO QUARTO INTERVALO, O NÓ 1 NO OITAVO INTERVALO E O NÓ 3 NO NONO INTERVALO

lítica probabilística de transmissão. Os únicos intervalos "não desperdiçados" serão aqueles em que exatamente um nó transmite. Um intervalo em que exatamente um nó transmite é denominado um **intervalo bem-sucedido**. A **eficiência** de um protocolo de acesso múltiplo com intervalos é definida como a fração (calculada durante um longo tempo) de intervalos bem-sucedidos no caso de haver grande número de nós ativos, cada qual tendo sempre grande número de quadros a enviar. Note que, se não fosse usado nenhum tipo de controle de acesso e cada nó retransmitisse logo após a colisão, a eficiência seria zero. É claro que o *slotted* ALOHA aumenta a eficiência para além de zero, mas em quanto?

Vamos agora esboçar a derivação da eficiência máxima do *slotted* ALOHA. Para manter a simplicidade dessa derivação, vamos modificar um pouco o protocolo e admitir que cada nó tenta transmitir um quadro em cada intervalo com probabilidade p. (Isto é, admitimos que cada nó sempre tenha um quadro para enviar, e transmita com probabilidade p tanto para um quadro novo como para um quadro que já sofreu uma colisão.) Suponha que haja N nós. Então, a probabilidade de que determinado intervalo seja um intervalo bem-sucedido é a probabilidade de que um dos nós transmita e os restantes $N - 1$ nós, não. A probabilidade de que determinado nó transmita é p; a de que os nós restantes não transmitam é $(1 - p)^{N-1}$. Por conseguinte, a probabilidade de que um dado nó tenha sucesso é $p(1 - p)^{N-1}$. Como há N nós, a probabilidade de um nó arbitrário ter sucesso é $Np(1 - p)^{N-1}$.

Assim, quando há N nós ativos, a eficiência do *slotted* ALOHA é $Np(1 - p)^{N-1}$. Para obtermos a eficiência *máxima* para N nós ativos, temos de encontrar um p^* que maximize essa expressão. (Veja os exercícios ao final deste capítulo para um esboço geral dessa derivação.) E, para obtermos a eficiência máxima para um grande número de nós ativos, consideramos o limite de $Np^*(1 - p^*)^{N-1}$ quando N tende ao infinito. (Novamente, veja os exercícios ao final deste capítulo.) Após esses cálculos, descobriremos que a eficiência máxima do protocolo é dada por $1/e = 0{,}37$. Isto é, quando um grande número de nós tem muitos quadros a transmitir, então (na melhor das hipóteses) apenas 37% dos intervalos realiza um trabalho útil. Assim, a taxa efetiva de transmissão do canal não é R bits/s, mas apenas 0,37 R bits/s! Uma análise semelhante também demonstra que 37% dos intervalos ficam vazios e 26% apresentam colisões. Imagine o pobre administrador de rede que comprou um sistema *slotted* ALOHA de 100 Mbits/s esperando poder usar a rede para transmitir dados entre um grande número de usuários a uma taxa agregada de, digamos, 80 Mbits/s! Embora o canal seja capaz de transmitir um dado quadro à taxa máxima do canal de 100 Mbits/s, no final, a vazão que se consegue com esse canal é de menos de 37 Mbits/s.

Aloha

O protocolo *slotted* ALOHA requer que todos os nós sincronizem suas transmissões para que comecem no início de um intervalo. O primeiro protocolo ALOHA [Abramson, 1970] era, na realidade, um protocolo sem intervalos e totalmente descentralizado. No ALOHA puro, quando um quadro chega pela primeira vez (isto é, um datagrama da camada de rede é passado para baixo a partir da camada de rede no nó remetente), o nó imediatamente transmite o quadro inteiro ao canal de difusão. Se um quadro transmitido sofrer uma colisão com uma ou mais transmissões, o nó retransmitirá de imediato (após ter concluído a transmissão total do quadro que sofreu a colisão) o quadro com probabilidade p. Caso contrário, o nó esperará por um tempo de transmissão de quadro. Após essa espera, ele então retransmite o quadro com probabilidade p ou espera (permanecendo ocioso) por outro tempo de quadro com probabilidade $1 - p$.

Para determinar a eficiência máxima do ALOHA puro, vamos focalizar um nó individual. Consideraremos as mesmas premissas que adotamos na análise do *slotted* ALOHA e tomaremos o tempo de transmissão do quadro como a unidade de tempo. A qualquer momento, a probabilidade de que um nó esteja transmitindo um quadro é p. Suponha que esse quadro comece a transmitir no tempo t_0. Como ilustra na Figura 5.11, para que ele seja transmitido com sucesso, nenhum dos outros nós pode começar sua transmissão no intervalo de tempo $[t_0 - 1, t_0]$. Esta se sobreporia ao início da transmissão do quadro do nó i. A probabilidade de que todos os outros nós não iniciem uma transmissão nesse intervalo é $(1 - p)^{N-1}$. De maneira semelhante, nenhum outro nó pode iniciar uma transmissão enquanto o nó i estiver transmitindo, pois essa transmissão se sobreporia à parte final da do nó i. A probabilidade de que todos os outros nós não iniciem uma transmissão nesse intervalo é também $(1 - p)^{N-1}$.

FIGURA 5.11 TRANSMISSÕES INTERFERENTES NO ALOHA PURO

Vai se sobrepor ao início do quadro de *i*

Vai se sobrepor ao final do quadro de *i*

Quadro do nó *i*

$t_0 - 1$ t_0 $t_0 + 1$ Tempo

Assim, a probabilidade de que um dado nó tenha uma transmissão bem-sucedida é $p(1 - p)^{2(N-1)}$. Levando ao limite, como fizemos no caso do *slotted* ALOHA, descobrimos que a eficiência máxima do protocolo ALOHA puro é de apenas $1/(2e)$ — exatamente a metade da eficiência do *slotted* ALOHA. É este o preço que se paga por um protocolo ALOHA totalmente descentralizado.

HISTÓRIA

Norm Abramsom e a ALOHAnet

Norm Abramsom, um doutor em engenharia, era apaixonado por surfe e interessado na comutação de pacotes. Essa combinação de interesses o levou à Universidade do Havaí em 1969. O Havaí é formado por muitas ilhas montanhosas, o que dificulta a instalação e a operação de redes terrestres. Quando não estava surfando, Abramson ficava pensando em como projetar uma rede que fizesse comutação de pacotes por rádio. A rede que ele projetou tinha um hospedeiro central e diversos nós secundários espalhados pelas ilhas havaianas. A rede tinha dois canais, cada um usando uma faixa de frequência diferente. O canal na direção dos nós secundários fazia difusão de pacotes do hospedeiro central para os secundários; e o canal na direção contrária enviava pacotes dos hospedeiros secundários para o central. Além de enviar pacotes de informação, o hospedeiro central também enviava pelo canal na direção dos nós secundários um reconhecimento para cada pacote recebido com sucesso dos secundários.

Como os hospedeiros secundários transmitiam pacotes de maneira descentralizada, inevitavelmente ocorriam colisões no canal entre eles e o hospedeiro central. Essa observação levou Abramson a inventar, em 1970, o protocolo ALOHA puro, descrito neste capítulo. Em 1970, com o financiamento contínuo da ARPA, ele conectou sua ALOHAnet à ARPAnet. O trabalho de Abramson é importante não somente porque foi o primeiro exemplo de uma rede de pacotes por rádio, mas também porque inspirou Bob Metcalfe. Alguns anos depois, Metcalfe modificou o protocolo ALOHA e criou o protocolo CSMA/CD e a rede local Ethernet.

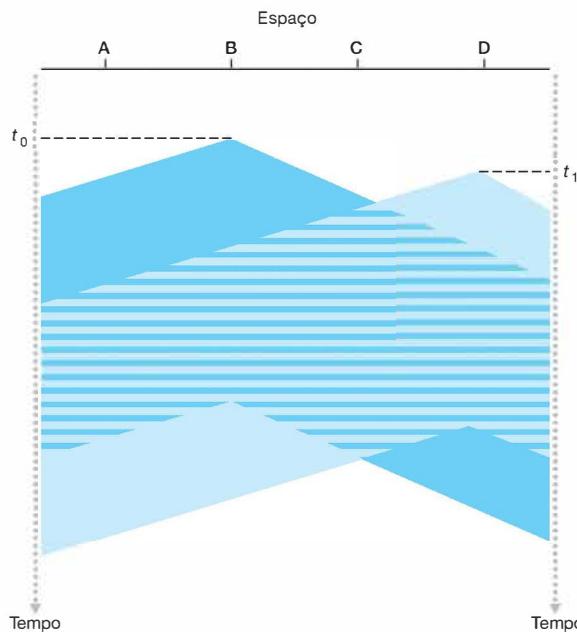
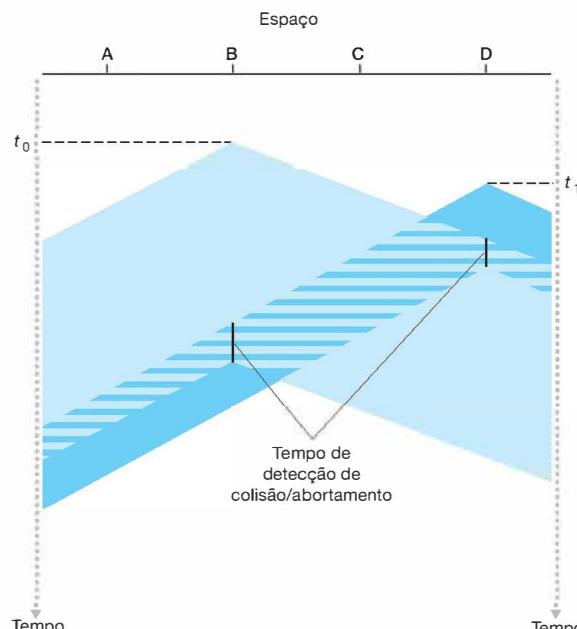
CSMA (acesso múltiplo com detecção de portadora)

Tanto no *slotted* ALOHA quanto no ALOHA puro, a decisão de transmitir é tomada por um nó independentemente da atividade dos outros nós ligados ao canal de difusão. Em particular, um nó não se preocupa se por acaso outro está transmitindo quando ele começa a transmitir nem para de transmitir se outro nó começar a interferir em sua transmissão. Em nossa analogia do coquetel, os protocolos ALOHA se parecem muito com um convidado mal-educado que continua a tagarelar mesmo quando outras pessoas estão falando. Como seres humanos, temos protocolos que nos levam não somente a nos comportar com mais civilidade, mas também a

reduzir o tempo que gastamos "colidindo" com outros durante a conversação e, por conseguinte, a aumentar a quantidade de dados que trocamos durante nossas conversas. Especificamente, há duas regras importantes que regem a conversação educada entre seres humanos:

- *Ouça antes de falar*. Se uma pessoa estiver falando, espere até que ela tenha terminado. No mundo das redes, isso é denominado **detecção de portadora** — um nó ouve o canal antes de transmitir. Se um quadro de outro nó estiver atualmente sendo transmitido para dentro do canal, o nó então esperará até que não detecte transmissões por um período de tempo curto, e então iniciará a transmissão.
- *Se alguém começar a falar ao mesmo tempo que você, pare de falar*. No mundo das redes, isso é denominado **detecção de colisão** — um nó que está transmitindo ouve o canal enquanto transmite. Se esse nó detectar que outro nó está transmitindo um quadro interferente, ele para de transmitir e espera por algum tempo antes de repetir o ciclo de detectar-e-transmitir-quando-ocioso.

Essas duas regras estão incorporadas na família de protocolos de **acesso múltiplo com detecção de portadora** (CSMA — *carrier sense multiple access*) e CSMA **com detecção de colisão** (CSMA/CD) [Kleinrock, 1975b; Metcalfe, 1976; Lam, 1980; Rom, 1990]. Foram propostas muitas variações do CSMA e do CSMA/CD. Nesta seção, consideraremos algumas das características mais importantes e fundamentais do CSMA e do CSMA/CD.

A primeira pergunta que se poderia fazer sobre o CSMA é a seguinte: se todos os nós realizam detecção de portadora, por que ocorrem colisões no primeiro lugar? Afinal, um nó vai abster-se de transmitir sempre que perceber que outro está transmitindo. A resposta a essa pergunta pode ser ilustrada utilizando diagramas espaço/tempo [Molle, 1987]. A Figura 5.12 apresenta um diagrama espaço/tempo de quatro nós (A, B, C, D) ligados a um barramento linear de transmissão. O eixo horizontal mostra a posição de cada nó no espaço; o eixo vertical representa o tempo.

No tempo t_0, o nó B percebe que o canal está ocioso, pois nenhum outro nó está transmitindo no momento. Assim, o nó B começa a transmitir e seus bits se propagam em ambas as direções ao longo do meio de transmissão. A propagação para baixo dos bits de B na Figura 5.12 com o aumento do tempo indica que é preciso uma quantidade de tempo de valor diferente de zero para que os bits de B de fato se propaguem (apesar de quase à velocidade da luz) ao longo do meio de transmissão. No tempo t_1 ($t_1 > t_0$), o nó D tem um quadro para enviar. Embora o nó B esteja transmitindo no tempo t_1, os bits que estão sendo transmitidos por B ainda não alcançaram D. Assim, D percebe o canal como ocioso em t_1. De acordo com o protocolo CSMA, D começa então a transmitir seu quadro. Pouco tempo depois, a transmissão de B passa a interferir na transmissão de D em D. Fica evidente, pela Figura 5.12, que o tempo de **atraso de propagação fim a fim de canal** para um canal de difusão — o tempo que leva para que um sinal se propague de um dos extremos do canal para outro — desempenhará um papel crucial na determinação de seu desempenho. Quanto mais longo for esse atraso de propagação, maior será a chance de um nó que detecta portadora ainda não poder perceber uma transmissão que já começou em outro nó da rede.

Carrier Sense Multiple Access with Collision Detection (CSMA/CD)

Na Figura 5.12, os nós não realizam detecção de colisão; ambos, B e D, continuam a transmitir seus quadros integralmente mesmo que ocorra uma colisão. Quando um nó realiza detecção de colisão, ele cessa a transmissão imediatamente. A Figura 5.13 mostra o mesmo cenário da Figura 5.12, exceto que cada um dos dois nós aborta sua transmissão pouco tempo após detectar uma colisão. É claro que adicionar detecção de colisão a um protocolo de acesso múltiplo ajudará o desempenho do protocolo por não transmitir inteiramente um quadro inútil, cujo conteúdo está corrompido (pela interferência de um quadro de outro nó).

Antes de analisar o protocolo CSMA/CD, vamos resumir sua operação do ponto de vista de um adaptador (em um nó) ligado a um canal de difusão:

1. O adaptador obtém um datagrama da camada de rede, prepara um quadro da camada de enlace e coloca o quadro no buffer do adaptador.

FIGURA 5.12 DIAGRAMA ESPAÇO/TEMPO DE DOIS NOS CSMA COM COLISÃO DE TRANSMISSÕES

FIGURA 5.13 CSMA COM DETECÇÃO DE COLISÃO

2. Se o adaptador detectar que o canal está ocioso (ou seja, não há energia de sinal entrando nele a partir do canal), ele começa a transmitir o quadro. Por outro lado, se detectar que o canal está ocupado, ele espera até que não detecte energia de sinal, para então começar a transmitir o quadro.
3. Enquanto transmite, o adaptador monitora a presença de energia de sinal vinda de outros adaptadores usando o canal de difusão.
4. Se transmitir o quadro inteiro sem detectar energia de sinal de outros adaptadores, o adaptador terá terminado com o quadro. Por outro lado, se detectar energia de sinal de outros adaptadores enquanto transmite, ele aborta a transmissão (ou seja, para de transmitir seu quadro).
5. Depois de abortar, o adaptador espera por um tempo aleatório e depois retorna à etapa 2.

Espera-se que a necessidade de esperar por um tempo aleatório (e não fixo) seja clara — se dois nós transmitissem quadros ao mesmo tempo e depois ambos esperassem pelo mesmo período de tempo fixo, eles continuariam colidindo indefinidamente. Mas qual é um bom intervalo de tempo para escolher um tempo de espera aleatório? Se o intervalo for grande e o número de nós colidindo for pequeno, é provável que os nós esperem muito tempo (com o canal permanecendo ocioso) antes de repetir a etapa de detectar-e-transmitir-quando-ocioso. Por outro lado, se o intervalo for pequeno e o número de nós colidindo for grande, é provável que os valores aleatórios escolhidos sejam quase os mesmos, e os nós transmitindo colidirão de novo. Gostaríamos de ter um intervalo que seja curto quando o número de nós colidindo for pequeno, porém longo quando for grande.

O algoritmo de **recuo exponencial binário**, usado na Ethernet e também nos protocolos de acesso múltiplo de rede a cabo DOCSIS [DOCSIS, 2011], resolve esse problema de forma elegante. Especificamente, ao transmitir um quadro que já tenha experimentado n colisões, um nó escolhe o valor de K aleatoriamente a partir de $\{0, 1, 2, \ldots, 2^n - 1\}$. Assim, quanto mais colisões um quadro experimentar, maior o intervalo do qual K é escolhido. Para Ethernet, a quantidade de tempo real que um nó recua é $K \cdot 512$ tempos de bit (isto é, K vezes a quantidade de tempo necessária para enviar 512 bits para a Ethernet) e o valor máximo que n pode tomar é limitado a 10.

Vejamos um exemplo. Suponha que um nó tente transmitir um quadro pela primeira vez e, enquanto transmite, ele detecta uma colisão. O nó, então, escolhe $K = 0$ com probabilidade 0,5 ou escolhe $K = 1$ com probabilidade 0,5. Se o nó escolhe $K = 0$, então ele de imediato começa a detectar o canal. Se o nó escolhe $K = 1$, ele espera 512 tempos de bit (por exemplo, 0,01 ms para uma Ethernet a 100 Mbits/s) antes de iniciar o ciclo de detectar-e-transmitir-quando-ocioso. Após uma segunda colisão, K é escolhido com probabilidade igual dentre {0,1,2,3}. Após três colisões, K é escolhido com probabilidade igual dentre {0,1,2,3,4,5,6,7}. Após dez ou mais colisões, K é escolhido com probabilidade igual dentre {0,1,2, . . . , 1023}. Assim, o tamanho dos conjuntos dos quais K é escolhido cresce exponencialmente com o número de colisões; por esse motivo, esse algoritmo é denominado recuo exponencial binário.

Observamos aqui também que, toda vez que um nó prepara um novo quadro para transmissão, ele roda o algoritmo CSMA/CD, não levando em conta quaisquer colisões que possam ter ocorrido no passado recente. Assim, é possível que um nó com um novo quadro possa escapar imediatamente em uma transmissão bem-sucedida enquanto vários outros nós estão no estado de recuo exponencial.

Eficiência do CSMA/CD

Quando somente um nó tem um quadro para enviar, esse nó pode transmitir na velocidade total do canal (por exemplo, para Ethernet, as velocidades típicas são 10 Mbits/s, 100 Mbits/s ou 1 Gbit/s). Porém, se muitos nós tiverem quadros para transmitir, a velocidade de transmissão eficaz do canal pode ser muito menor. Definimos a **eficiência do CSMA/CD** como a fração de tempo (por um período longo) durante a qual os quadros estão sendo transmitidos no canal sem colisões quando há um grande número de nós ativos, com cada nó tendo um grande número de quadros para enviar. Para apresentar uma aproximação fechada da eficiência da Ethernet, considere que d_{prop} indica o tempo máximo que uma energia de sinal leva para ser

propagada entre dois adaptadores quaisquer. Considere que d_{trans} seja o tempo para transmitir um quadro de tamanho máximo (cerca de 1,2 ms para uma Ethernet a 10 Mbits/s). Uma derivação da eficiência do CSMA/CD está fora do escopo deste livro (ver Lam [1980] e Bertsekas [1991]). Aqui, indicamos simplesmente a seguinte aproximação:

$$\text{Eficiência} = \frac{1}{1 + 5d_{prop}/d_{trans}}$$

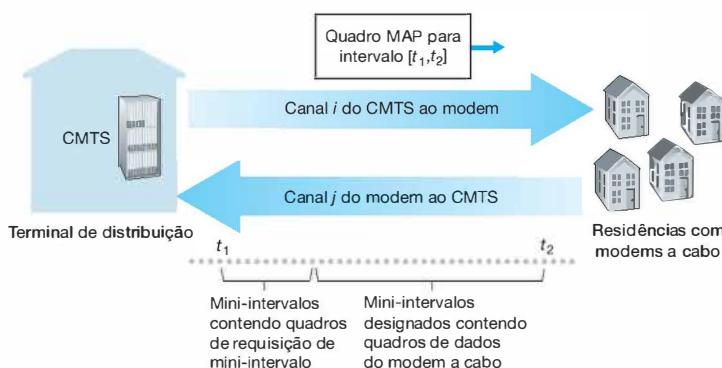
Por esta fórmula, vemos que, à medida que d_{prop} tende a 0, a eficiência tende a 1. Isso corresponde à intuição de que, se o atraso de propagação for zero, os nós colidindo serão imediatamente abortados, sem desperdiçar o canal. Além disso, à medida que d_{trans} se torna muito grande, a eficiência tende a 1. Isso também é intuitivo, pois, quando um quadro agarra o canal, prende-se a ele por um longo tempo; assim, o canal estará realizando trabalho produtivo por quase todo o tempo.

5.3.3 Protocolos de revezamento

Lembre-se de que duas propriedades desejáveis de um protocolo de acesso múltiplo são: (1) quando apenas um nó está ativo, este tem uma vazão de R bits/s; (2) quando M nós estão ativos, então cada nó ativo tem uma vazão de mais ou menos R/M bits/s. Os protocolos ALOHA e CSMA têm a primeira propriedade, mas não a segunda. Isso motivou os pesquisadores a criarem outra classe de protocolos — os **protocolos de revezamento**. Como acontece com os de acesso aleatório, há dezenas de protocolos de revezamento, e cada um deles tem muitas variações. Discutiremos nesta seção dois dos mais importantes. O primeiro é o **protocolo de *polling*** (seleção). Ele requer que um dos nós seja designado como nó mestre. Este **seleciona** cada um dos nós por alternância circular. Em particular, ele envia primeiro uma mensagem ao nó 1 dizendo que ele (o nó 1) pode transmitir até certo número máximo de quadros. Após o nó 1 transmitir alguns quadros, o nó mestre diz ao nó 2 que ele (o nó 2) pode transmitir até certo número máximo de quadros. (O nó mestre pode determinar quando um nó terminou de enviar seus quadros observando a ausência de um sinal no canal.) O procedimento continua dessa maneira, com o nó mestre escolhendo cada um dos nós de maneira cíclica.

O protocolo de *polling* elimina as colisões e os intervalos vazios que atormentam os protocolos de acesso aleatório, e isso permite que ele tenha uma eficiência muito maior. Mas também tem algumas desvantagens. A primeira é que o protocolo introduz um atraso de seleção — o período de tempo exigido para notificar um nó que ele pode transmitir. Se, por exemplo, apenas um nó estiver ativo, então ele transmitirá a uma velocidade menor do que R bits/s, pois o nó mestre tem de escolher cada um dos nós ociosos por vez, cada vez que um nó ativo tiver enviado seu número máximo de quadros. A segunda desvantagem é potencialmente mais séria: se o nó mestre falhar, o canal inteiro ficará inoperante. O Protocolo 802.15 e o protocolo Bluetooth, que estudaremos na Seção 6.3, são exemplos de protocolos de *polling*.

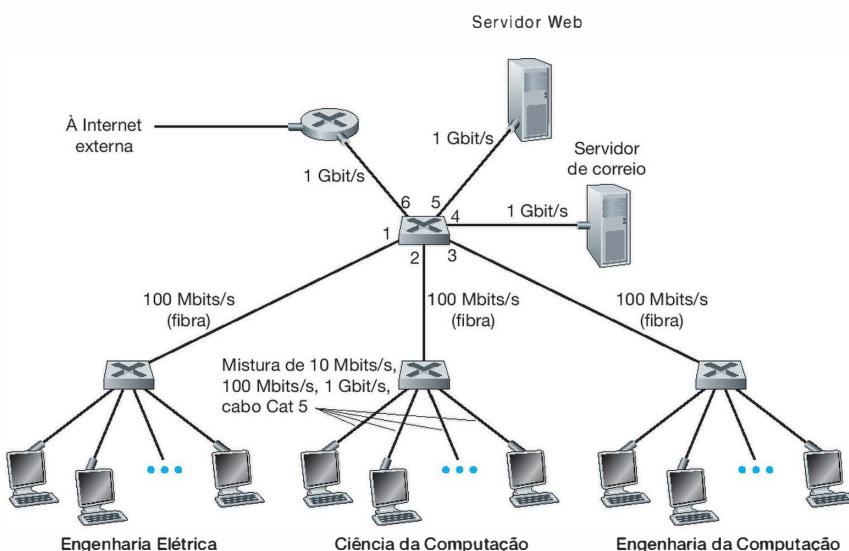
O segundo protocolo de revezamento é o **protocolo de passagem de permissão**. Nele, não há nó mestre. Um pequeno quadro de finalidade especial conhecido como uma **permissão** (*token*) é passado entre os nós obedecendo a uma determinada ordem fixa. Por exemplo, o nó 1 poderá sempre enviar a permissão ao nó 2, o nó 2 poderá sempre enviar a permissão ao nó 3, o nó N poderá sempre enviar a permissão ao nó 1. Quando um nó recebe uma permissão, ele a retém apenas se tiver alguns quadros para transferir, caso contrário, imediatamente a repassa para o nó seguinte. Se um nó tiver quadros para transmitir quando recebe a permissão, ele enviará um número máximo de quadros e, em seguida, passará a permissão para o nó seguinte. A passagem de permissão é descentralizada e tem uma alta eficiência. Mas também tem seus problemas. Por exemplo, a falha de um nó pode derrubar o canal inteiro. Ou, se um nó acidentalmente se descuida e não libera a permissão, então é preciso chamar algum procedimento de recuperação para recolocar a permissão em circulação. Durante muitos anos, foram desenvolvidos muitos protocolos de passagem de permissão, e cada um deles teve de enfrentar esses e outros assuntos delicados, incluindo o protocolo FDDI (*Fiber Distributed Data Interface*) [Jain, 1994] e o protocolo *token ring* IEEE 802.5 [IEEE 802.5, 2012].

5.3.4 DOCSIS: o protocolo da camada de enlace para acesso à Internet a cabo

Nas três subseções anteriores, aprendemos a respeito de três classes gerais de protocolos de acesso múltiplo: protocolos de divisão de canal, protocolos de acesso aleatório e protocolos de revezamento. Aqui, uma rede de acesso a cabo servirá como um excelente estudo de caso, pois veremos aspectos de *cada uma* dessas três classes de protocolos de acesso múltiplo com a rede de acesso a cabo!

Lembre-se de que, na Seção 1.2.1, vimos que uma rede de acesso a cabo em geral conecta milhares de modems a cabo residenciais a um sistema de término de modem a cabo (CMTS) no terminal de distribuição da rede a cabo. Data-Over-Cable Service Interface Specifications (DOCSIS) [DOCSIS, 2011] especifica a arquitetura de rede de dados a cabo e seus protocolos. DOCSIS utiliza FDM para dividir os segmentos de rede em direção ao modem (*downstream*) e em direção ao CMTS (*upstream*) em canais de múltiplas frequências. Cada canal do CMTS ao modem tem 6 MHz de largura, com uma vazão máxima de cerca de 40 Mbits/s por canal (embora, na prática, essa velocidade de dados raramente seja vista em um modem a cabo); cada canal do modem ao CMTS tem uma largura de canal máxima de 6,4 MHz, e uma vazão máxima de mais ou menos 30 Mbits/s. Cada um deles é um canal de difusão. Os quadros transmitidos no canal do CMTS ao modem são recebidos por todos os modems a cabo que recebem esse canal; porém, como há apenas um CMTS transmitindo para o canal em direção ao modem, não há problema de acesso múltiplo. A direção contrária, no entanto, é mais interessante e tecnicamente desafiadora, pois vários modems a cabo compartilham o mesmo canal (frequência) em direção ao CMTS, e com isso potencialmente haverá colisões.

Conforme ilustra a Figura 5.14, cada canal do modem ao CMTS é dividido em intervalos de tempo (tipo TDM), cada um com uma sequência de mini-intervalos, durante os quais os modems a cabo podem transmitir ao CMTS. O CMTS concede permissão explicitamente aos modems individuais para transmitir durante mini-intervalos específicos. O CMTS realiza isso enviando uma mensagem de controle (conhecida como mensagem MAP) em um canal em direção ao modem, para especificar qual modem a cabo (com dados para enviar) pode transmitir durante qual mini-intervalo durante o tempo especificado na mensagem de controle. Como os mini-intervalos são alocados explicitamente aos modems a cabo, o CMTS pode garantir que não haverá transmissões colidindo durante um mini-intervalo.

Mas como o CMTS sabe quais modems a cabo possuem dados para enviar em primeiro lugar? Isso é feito pelo envio de quadros de requisição de mini-intervalo dos modems ao CMTS, durante um conjunto especial de mini-intervalos dedicados para essa finalidade, como mostra a Figura 5.14. Esses quadros de requisição de mini-intervalo são transmitidos em uma forma de acesso aleatório e, portanto, podem colidir uns com os outros. Um modem a cabo não consegue detectar se o canal até o CMTS está ocupado nem detectar colisões.

FIGURA 5.14 CANAIS ENTRE O CMTS E OS MODEMS A CABO

Em vez disso, ele deduz que seu quadro de requisição de mini-intervalo teve uma colisão se não receber uma resposta à alocação requisitada na próxima mensagem de controle do CMTS ao modem. Ao deduzir uma colisão, um modem a cabo usa o recuo exponencial binário para adiar a retransmissão do seu quadro de requisição de mini-intervalo para um período de tempo no futuro. Quando há pouco tráfego no canal de subida até o CMTS, um modem a cabo pode de fato transmitir quadros de dados durante os intervalos designados nominalmente para os quadros de requisição de mini-intervalo (evitando, assim, ter que esperar por uma designação de mini-intervalo).

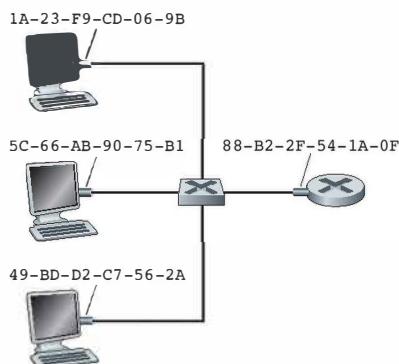
Uma rede de acesso a cabo, portanto, serve como um excelente exemplo de protocolos de acesso múltiplo em ação — FDM, TDM, acesso aleatório e intervalos de tempo alocados de forma central, tudo dentro de uma rede!

5.4 REDES LOCAIS COMUTADAS

Tendo explicado as redes de difusão e os protocolos de acesso múltiplo na seção anterior, vamos voltar nossa atenção em seguida às redes locais comutadas. A Figura 5.15 mostra uma rede local comutada conectando três departamentos, dois servidores e um roteador com quatro comutadores. Como esses comutadores operam na camada de enlace, eles comutam quadros da camada de enlace (em vez de datagramas da camada de rede), não reconhecem endereços da camada de rede e não utilizam algoritmos de roteamento, como RIP ou OSPF, para determinar caminhos pela rede de comutadores da camada 2. Em vez de usar endereços IP, logo veremos que eles usam endereços da camada de enlace para repassar quadros da camada de enlace pela rede de comutadores. Vamos começar nosso estudo das LANs comutadas explicando primeiro o endereçamento na camada de enlace (Seção 5.4.1). Depois, examinaremos o famoso protocolo Ethernet (Seção 5.4.2). Depois de examinar o endereçamento na camada de enlace e Ethernet, veremos como operam os comutadores da camada de enlace (Seção 5.4.3) e depois (na Seção 5.4.4) como esses comutadores normalmente são usados para montar LANs em grande escala.

FIGURA 5.15 UMA REDE INSTITUCIONAL CONECTADA POR QUATRO COMUTADORES

Servidor Web
À Internet externa
1 Gbit/s
1 Gbit/s
Servidor de correio
1 Gbit/s
6 5 4 1 2 3
100 Mbits/s (fibra)
100 Mbits/s (fibra)
100 Mbits/s (fibra)
Mistura de 10 Mbits/s, 100 Mbits/s, 1 Gbit/s, cabo Cat 5
Engenharia Elétrica
Ciência da Computação
Engenharia da Computação

5.4.1 Endereçamento na camada de enlace e ARP

Hospedeiros e roteadores têm endereços da camada de enlace. Você talvez fique surpreso com isso ao lembrar que dissemos, no Capítulo 4, que nós também têm endereços da camada de rede e talvez se pergunte por que, afinal de contas, é preciso ter endereços na camada de rede e na de enlace. Nesta seção, além de descrevermos a sintaxe e a função dos endereços da camada de enlace, esperamos esclarecer por que as duas camadas de endereços são úteis e, na verdade, indispensáveis. Estudaremos também o Protocolo de Resolução de Endereços (*Address Resolution Protocol* — ARP), que oferece um mecanismo que habilita os nós a traduzirem endereços IP para endereços da camada de enlace.

Endereços MAC

Na verdade, não é o nó (isto é, o hospedeiro ou o roteador) que tem um endereço da camada de enlace, mas o adaptador do nó. Um hospedeiro ou roteador com várias interfaces de rede, portanto, terá vários endereços da camada de enlace associados a ele, assim como também teria vários endereços IP associados. Porém, é importante observar que os comutadores da camada de enlace não têm endereços da camada de enlace associados às suas interfaces, que se conectam aos hospedeiros e roteadores. Isso porque a função do comutador da camada de enlace é transportar datagramas entre hospedeiros e roteadores; um comutador faz isso de modo transparente, ou seja, sem que o hospedeiro ou roteador tenha que endereçar o quadro explicitamente para o comutador intermediário. Isso é ilustrado na Figura 5.16. Um endereço da camada de enlace é também denominado **endereço de LAN**, **endereço físico** ou **endereço MAC** (Media Access Control — controle de acesso ao meio). Como a expressão endereço MAC parece ser a mais popular, daqui em diante nos referiremos a endereços da camada de enlace como endereços MAC. Para a maior parte das LANs (incluindo a Ethernet e as LANs 802.11 sem fio), o endereço MAC tem 6 bytes de comprimento, o que dá 2^{48} endereços MAC possíveis. Como ilustrado na Figura 5.16, tais endereços de 6 bytes costumam ser expressos em notação hexadecimal, com cada byte do endereço mostrado como um par de números hexadecimais. Apesar de os endereços MAC serem projetados como permanentes, agora é possível mudar o endereço de MAC de um adaptador via software. No entanto, pelo resto desta seção, vamos considerar que o endereço MAC de um adaptador é fixo.

Uma propriedade interessante dos endereços MAC é que não existem dois adaptadores com o mesmo endereço. Isso pode parecer surpreendente, dado que os adaptadores são fabricados em muitos países por inúmeras empresas diferentes. Como uma empresa fabricante de adaptadores em Taiwan se certifica de que está usando endereços diferentes dos usados por um fabricante de adaptadores na Bélgica? A resposta é que o IEEE gerencia o espaço físico de endereços MAC. Em particular, quando uma empresa quer produzir adaptadores, compra, por

FIGURA 5.16 CADA INTERFACE CONECTADA À LAN TEM UM ENDEREÇO MAC EXCLUSIVO

uma taxa nominal, uma parcela do espaço de endereços que consiste em 2^{24} endereços. O IEEE aloca a parcela de 2^{24} endereços fixando os primeiros 24 bits de um endereço MAC e permitindo que a empresa crie combinações exclusivas com os últimos 24 bits para cada adaptador.

O endereço MAC de um adaptador tem uma estrutura linear (oposta à estrutura hierárquica) e nunca muda, não importando para onde vá o adaptador. Um notebook com um cartão Ethernet tem sempre o mesmo endereço MAC, não importando aonde o computador vá. Um smartphone com uma interface 802.11 tem sempre o mesmo endereço MAC aonde quer que vá. Lembre-se de que, ao contrário, um endereço IP tem uma estrutura hierárquica (isto é, uma parte que é da rede e outra que é do hospedeiro) e que o endereço IP de um nó precisa ser trocado quando o hospedeiro troca de lugar, por exemplo, muda a rede à qual está conectado. O endereço MAC de um adaptador é semelhante ao número do CPF de uma pessoa, que também tem uma estrutura linear e não muda, não importando aonde a pessoa vá. Um endereço IP é semelhante ao endereço postal de uma pessoa, que é hierárquico e precisa ser trocado quando a pessoa muda de endereço. Exatamente como uma pessoa pode achar útil ter um endereço postal, bem como um número de CPF, também é útil para um nó ter um endereço da camada de rede, assim como um endereço MAC.

Quando um adaptador quer enviar um quadro para algum adaptador de destino, o remetente insere no quadro o endereço MAC do destino e envia o quadro para dentro da LAN. Como veremos em breve, um comutador às vezes transmite por difusão um quadro que chega para todas as suas interfaces. Veremos, no Capítulo 6, que a LAN 802.11 também transmite quadros por difusão. Assim, um adaptador pode receber um quadro que não está endereçado a ele. Desse modo, quando o adaptador receber um quadro, ele verificará se o endereço MAC de destino combina com seu próprio endereço MAC. Se ambos combinarem, o adaptador extrairá o datagrama encerrado no quadro e o passará para cima na pilha de protocolos. Se não combinarem, o adaptador descartará o quadro sem passar o datagrama da camada de rede para cima na pilha de protocolos. Assim, somente o destino será interrompido quando receber um quadro.

No entanto, às vezes um adaptador remetente *quer* que todos os outros adaptadores na LAN recebam e *processem* o quadro que ele está prestes a enviar. Nesse caso, o adaptador remetente insere **um endereço de difusão** MAC especial no campo de endereço do destinatário do quadro. Para LANs que usam endereços de 6 bytes (como a Ethernet e 802.11), o endereço de difusão é uma cadeia de 48 bits 1 consecutivos (isto é, FF-FF-FF-FF-FF-FF em notação hexadecimal).

PRINCÍPIOS NA PRÁTICA

Mantendo a independência das camadas

Há diversas razões por que os nós têm endereços MAC além de endereços da camada de rede. Primeiro, LANs são projetadas para protocolos da camada de rede arbitrários, e não apenas para IP e para a Internet. Se os adaptadores recebessem endereços IP, e não os endereços MAC "neutros", eles não poderiam suportar com facilidade outros protocolos da camada de rede (por exemplo, IPX ou DECNet). Segundo, se adaptadores usassem endereços da camada de rede — em vez de endereços MAC — o endereço da camada de rede teria de ser armazenado na RAM do adaptador e reconfigurado toda vez que este mudasse de local (ou fosse ligado). Outra opção é não usar nenhum endereço nos adaptadores e fazer que cada um deles passe os dados (em geral, um datagrama IP) de cada quadro que recebe para cima na pilha de protocolos. A camada de rede poderia, então, verificar se o endereço combina com o da camada de rede. Um problema com essa opção é que o hospedeiro seria interrompido por cada quadro enviado à LAN, inclusive pelos destinados a outros nós na mesma LAN de difusão. Em resumo, para que as camadas sejam blocos de construção praticamente independentes em uma arquitetura de rede, diferentes camadas precisam ter seu próprio esquema de endereçamento. Já vimos até agora três tipos diferentes de endereços: nomes de hospedeiros para a camada de aplicação, endereços IP para a camada de rede e endereços MAC para a camada de enlace.

ARP (protocolo de resolução de endereços)

Como existem endereços da camada de rede (por exemplo, IP da Internet) e da camada de enlace (isto é, endereços MAC), é preciso fazer a tradução de um para o outro. Para a Internet, esta é uma tarefa do **protocolo de resolução de endereços** (Address Resolution Protocol — **ARP**) [RFC 826].

Para compreender a necessidade de um protocolo como o ARP, considere a rede mostrada na Figura 5.17. Nesse exemplo simples, cada hospedeiro e roteador tem um único endereço IP e um único endereço MAC. Como sempre, endereços IP são mostrados em notação decimal com pontos e endereços MAC, em notação hexadecimal. Para os propósitos desta discussão, vamos considerar nesta seção que o comutador transmite todos os quadros por difusão; isto é, sempre que um comutador recebe um quadro em uma interface, ele o repassa para todas as suas outras interfaces. Na próxima seção, vamos dar uma explicação mais precisa sobre como os comutadores trabalham.

Agora, suponha que o nó com endereço IP 222.222.222.220 queira mandar um datagrama IP para o nó 222.222.222.222. Nesse exemplo, os nós de origem e de destino estão na mesma sub-rede, no sentido do endereçamento estudado na Seção 4.4.2. Para enviar um datagrama, o nó de origem deve dar a seu adaptador não somente o datagrama IP, mas também o endereço MAC para o nó de destino 222.222.222.222. O adaptador do nó remetente montará então um quadro da camada de enlace contendo o endereço MAC do nó receptor e enviará o quadro para a LAN.

A pergunta importante considerada nesta seção é: como o nó remetente determina o endereço MAC para o nó com endereço IP 222.222.222.222? Como você já deve ter adivinhado, ele usa o ARP. Um módulo ARP no nó remetente toma como entrada qualquer endereço IP na mesma LAN e retorna o endereço MAC correspondente. Nesse exemplo, o nó remetente 222.222.222.220 fornece a seu módulo ARP o endereço IP 222.222.222.222 e o módulo ARP retorna o endereço MAC correspondente, 49-BD-D2-C7-56-2A.

Assim, vemos que o ARP converte um endereço IP para um endereço MAC. Em muitos aspectos, o ARP é semelhante ao DNS (estudado na Seção 2.5), que converte nomes de hospedeiros para endereços IP. Contudo, uma importante diferença entre os dois conversores é que o DNS converte nomes de hospedeiros para máquinas em qualquer lugar da Internet, ao passo que o ARP converte endereços IP apenas para nós na mesma sub-rede. Se um nó na Califórnia tentasse usar o ARP para converter o endereço IP de um nó no Mississippi, o ARP devolveria um erro.

Agora que já explicamos o que o ARP faz, vamos ver como ele funciona. Cada nó (hospedeiro ou roteador) tem em sua RAM uma **tabela ARP** que contém mapeamentos de endereços IP para endereços MAC. A Figura 5.18 mostra como seria uma tabela ARP no nó 222.222.222.220. Essa tabela também contém um valor de tempo de vida (TTL) que indica quando cada mapeamento será apagado. Note que a tabela não contém necessariamente um registro para cada hospedeiro da sub-rede; alguns podem jamais ter sido registrados, enquanto outros podem ter expirado. Um tempo de remoção típico para um registro é de 20 minutos a partir do momento em que foi colocado em uma tabela ARP.

FIGURA 5.17 CADA INTERFACE EM UMA LAN TEM UM ENDEREÇO IP E UM ENDEREÇO MAC

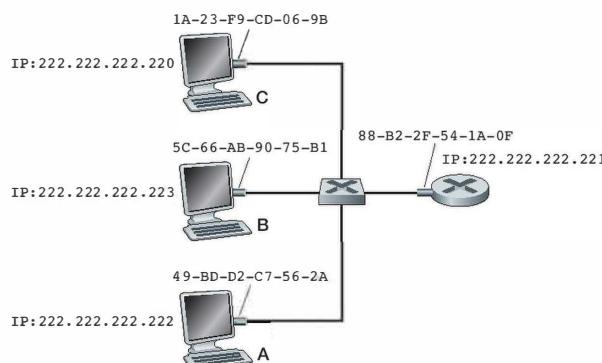



FIGURA 5.18 UMA POSSÍVEL TABELA ARP NO NÓ 222.222.222.220

Endereço IP	Endereço MAC	TTL
222.222.222.221	88-B2-2F-54-1A-0F	13:45:00
222.222.222.223	5C-66-AB-90-75-B1	13:52:00

Suponha agora que o hospedeiro 222.222.222.220 queira enviar um datagrama que tem endereço IP para outro nó daquela sub-rede. O nó remetente precisa obter o endereço MAC do nó de destino, dado o endereço IP daquele mesmo nó. Essa tarefa será fácil se a tabela ARP do nó remetente tiver um registro para esse nó de destino. Mas, e se, naquele momento, a tabela ARP não tiver um registro para o destinatário? Em particular, suponha que 222.222.222.220 queira enviar um datagrama para 222.222.222.222. Nesse caso, o remetente usa o protocolo ARP para converter o endereço. Primeiro, monta um pacote especial denominado **pacote ARP**. Um pacote ARP tem diversos campos, incluindo os endereços IP e MAC de envio e de recepção. Os pacotes ARP de consulta e de resposta têm o mesmo formato. A finalidade do pacote de consulta ARP é pesquisar todos os outros hospedeiros e roteadores na sub-rede para determinar o endereço MAC correspondente ao endereço IP que está sendo convertido.

Voltando a nosso exemplo, o nó 222.222.222.220 passa um pacote de consulta ARP ao adaptador junto com uma indicação de que este deve enviar o pacote ao endereço MAC de difusão, a saber, FF-FF-FF-FF-FF-FF. O adaptador encapsula o pacote ARP em um quadro da camada de enlace, usa o endereço de difusão como endereço de destino do quadro e transmite o quadro para a sub-rede. Retomando nossa analogia de número do CPF/endereço postal, note que a consulta ARP equivale a uma pessoa gritar em uma sala cheia de baias em alguma empresa (digamos, a empresa AnyCorp): "Qual é o número do CPF da pessoa cujo endereço é Baia 13, Sala 112, AnyCorp, Palo Alto, Califórnia?". O quadro que contém a consulta ARP é recebido por todos os outros adaptadores na sub-rede e (por causa do endereço de difusão) cada adaptador passa o pacote ARP dentro do quadro para o seu módulo ARP. Cada um desses módulos ARP verifica se seu endereço IP corresponde ao endereço IP de destino no pacote ARP. O único nó que atende a essa condição devolve um pacote ARP de resposta ao hospedeiro que fez a consulta, com o mapeamento desejado. O hospedeiro que fez a consulta (222.222.222.220) pode, então, atualizar sua tabela ARP e enviar seu datagrama IP, revestido com um quadro da camada de enlace, cujo endereço MAC de destino é aquele do hospedeiro ou roteador que respondeu à consulta ARP anterior.

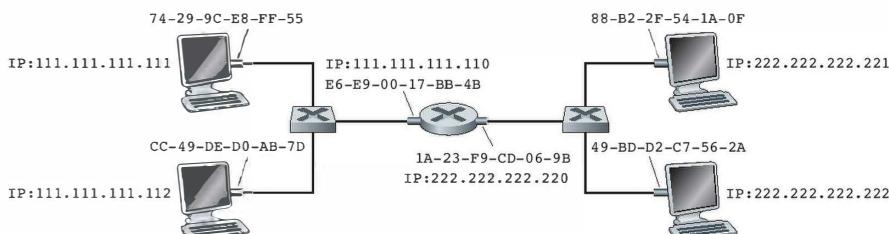
O protocolo ARP apresenta algumas características interessantes. Primeiro, a mensagem de consulta ARP é enviada dentro de um quadro de difusão, ao passo que a mensagem de resposta ARP é enviada dentro de um quadro padrão. Antes de continuar a leitura, é bom que você pense por que isso acontece. Segundo, o ARP é do tipo *plug-and-play*, isto é, a tabela de um nó ARP é construída automaticamente — ela não tem de ser configurada por um administrador de sistemas. E, se um nó for desligado da sub-rede, seu registro será enfim apagado das outras tabelas ARP na sub-rede.

Estudantes se perguntam se o ARP é como um protocolo da camada de enlace ou um protocolo da camada de rede. Como vimos, um pacote ARP é encapsulado dentro de um quadro da camada de enlace e, assim, encontra-se acima da camada de enlace, do ponto de vista da arquitetura. No entanto, um pacote ARP tem campos que contêm endereços da camada de rede, dessa forma é também comprovadamente um protocolo da camada de rede. Em suma, o ARP é talvez mais bem considerado um protocolo que fica em cima do limite entre as camadas de enlace e de rede — não se adequando perfeitamente na simples pilha de protocolos que estudamos no Capítulo 1. Os protocolos do mundo real têm complexidades desse tipo!

Envio de um datagrama para fora da sub-rede

Agora já deve estar claro como o ARP funciona quando um nó quer enviar um datagrama a outro nó *na mesma sub-rede*. Mas vamos examinar uma situação mais complicada, em que um hospedeiro de uma sub-rede

quer enviar um datagrama da camada de rede para um nó *que está fora da sub-rede* (isto é, passa por um roteador e entra em outra sub-rede). Vamos discutir essa questão no contexto da Figura 5.19, que mostra uma rede simples constituída de duas sub-redes interconectadas por um roteador.

Há diversos pontos interessantes a notar na Figura 5.19. Cada hospedeiro tem exatamente um endereço IP e um adaptador. Mas, como vimos no Capítulo 4, um roteador tem um endereço IP para *cada* uma de suas interfaces. Para cada interface de roteador também há um módulo ARP (dentro do roteador) e um adaptador. Como o roteador da Figura 5.19 tem duas interfaces, ele tem dois endereços IP, dois módulos ARP e dois adaptadores. É claro que cada adaptador na rede tem seu próprio endereço MAC.

Note também que a Sub-rede 1 tem endereços de rede 111.111.111/24 e que a Sub-rede 2 tem endereços de rede 222.222.222/24. Assim, todas as interfaces conectadas à Sub-rede 1 têm o formato 111.111.111.xxx e todas as conectadas à Sub-rede 2 têm o formato 222.222.222.xxx.

Agora, vamos examinar como um hospedeiro na Sub-rede 1 enviaria um datagrama a um hospedeiro na Sub-rede 2. Especificamente, suponha que o hospedeiro 111.111.111.111 queira enviar um datagrama IP ao hospedeiro 222.222.222.222. O hospedeiro remetente passa o datagrama a seu adaptador, como sempre. Mas ele deve também indicar a seu adaptador um endereço MAC de destino apropriado. E que endereço MAC o adaptador deveria usar? Poderíamos arriscar o palpite de que é aquele do adaptador do hospedeiro 222.222.222.222, a saber, 49-BD-D2-C7-56-2A. Mas esse palpite estaria errado! Se o adaptador remetente usasse aquele endereço MAC, nenhum dos adaptadores da Sub-rede 1 se preocuparia em passar os datagramas IP para cima, para sua camada de rede, já que o endereço de destino do quadro não combinaria com o endereço MAC de nenhum adaptador na Sub-rede 1. O datagrama apenas morreria e iria para o céu dos datagramas.

Se examinarmos cuidadosamente a Figura 5.19, veremos que, para um datagrama ir de 111.111.111.111 até um nó da Sub-rede 2, ele teria de ser enviado primeiro à interface de roteador 111.111.111.110, que é o endereço IP do roteador do primeiro salto no caminho até o destino final. Assim, o endereço MAC apropriado para o quadro é o endereço do adaptador para a interface de roteador 111.111.111.110, a saber, E6-E9-00-17-BB-4B. Como o hospedeiro remetente consegue o endereço MAC para a interface 111.111.111.110? Usando o ARP, é claro! Tão logo tenha esse endereço MAC, o adaptador remetente cria um quadro (contendo o datagrama endereçado para 222.222.222.222) e o envia para a Sub-rede 1. O adaptador do roteador na Sub-rede 1 verifica que o quadro da camada de enlace está endereçado a ele e, por conseguinte, o passa para a camada de rede do roteador. Viva! O datagrama IP foi transportado com sucesso do hospedeiro de origem para o roteador! Mas não acabamos. Ainda temos de levar o datagrama do roteador até o destino. O roteador agora tem de determinar a interface correta para a qual o datagrama deve ser repassado. Como discutimos no Capítulo 4, isso é feito pela consulta a uma tabela de repasse no roteador. A tabela de repasse indica o roteador para o qual o datagrama deve ser repassado via interface de roteador 222.222.222.220. Essa interface, então, passa o datagrama a seu adaptador, que o encapsula em um novo quadro e envia o quadro para a Sub-rede 2. Dessa vez, o endereço MAC de destino do quadro é, na verdade, o endereço MAC do destino final. E de onde o roteador obtém esse endereço MAC de destino? Do ARP, é claro!

O ARP para Ethernet está definido no RFC 826. Uma boa introdução ao ARP é dada no tutorial do TCP/IP, RFC 1180. Exploraremos o ARP mais detalhadamente nos exercícios ao final deste capítulo.

FIGURA 5.19 DUAS SUB-REDES INTERCONECTADAS POR UM ROTEADOR

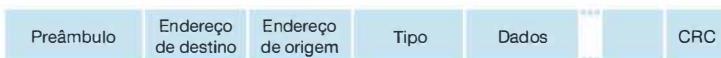
74-29-9C-E8-FF-55
IP:111.111.111.111
CC-49-DE-D0-AB-7D
IP:111.111.111.112
IP:111.111.111.110
E6-E9-00-17-BB-4B
1A-23-F9-CD-06-9B
IP:222.222.222.220
88-B2-2F-54-1A-0F
IP:222.222.222.221
49-BD-D2-C7-56-2A
IP:222.222.222.222

5.4.2 Ethernet

A Ethernet praticamente tomou conta do mercado de LANs com fio. Na década de 1980 e início da década de 1990, ela enfrentou muitos desafios de outras tecnologias LAN, incluindo *token ring*, FDDI e ATM. Algumas dessas outras tecnologias conseguiram conquistar uma parte do mercado de LANs durante alguns anos. Mas, desde sua invenção, em meados da década de 1970, a Ethernet continuou a se desenvolver e crescer e conservou sua posição dominante no mercado. Hoje, é de longe a tecnologia preponderante de LAN com fio e deve continuar assim no futuro previsível. Podemos dizer que a Ethernet está sendo para a rede local o que a Internet tem sido para a rede global.

Há muitas razões para o sucesso da Ethernet. Primeiro, ela foi a primeira LAN de alta velocidade amplamente disseminada. Como foi disponibilizada cedo, os administradores de rede ficaram bastante familiarizados com a Ethernet — com suas maravilhas e sutilezas — e relutaram em mudar para outras tecnologias LAN quando estas apareceram em cena. Segundo, *token ring*, FDDI e ATM são tecnologias mais complexas e mais caras do que a Ethernet, o que desencorajou ainda mais os administradores na questão da mudança. Terceiro, a razão mais atraente para mudar para uma outra tecnologia LAN (como FDDI e ATM) era em geral a velocidade mais alta da nova tecnologia; contudo, a Ethernet sempre se defendeu produzindo versões que funcionavam a velocidades iguais, ou mais altas. E, também, a Ethernet comutada foi introduzida no início da década de 1990, o que aumentou ainda mais suas velocidades efetivas de dados. Por fim, como se tornou muito popular, o hardware para Ethernet (em particular, adaptadores e comutadores) passou a ser mercadoria comum, de custo muito baixo.

A LAN Ethernet original foi inventada em meados da década de 1970 por Bob Metcalfe e David Boggs, e usava um barramento coaxial para interconectar os nós. As topologias de barramento da Ethernet persistiram durante toda a década de 1980 e até metade da década de 1990. Com uma topologia de barramento, a Ethernet é uma LAN de transmissão por difusão — todos os quadros transmitidos movem-se para, e são processados por, *todos* os adaptadores conectados ao barramento. Lembre-se de que vimos o protocolo de acesso múltiplo CSMA/CD da Ethernet com o recuo exponencial binário na Seção 5.3.2.

No fim da década de 1990, a maioria das empresas e universidades já tinha substituído suas LANs por instalações Ethernet usando topologia de estrela baseada em um *hub* (repetidor). Nessas instalações, os hospedeiros (e roteadores) estão diretamente conectados a um *hub* com um cabo de pares trançados de cobre. Um ***hub*** é um dispositivo de camada física que atua sobre bits individuais e não sobre quadros. Quando um bit, representando 0 ou 1, chega de uma interface, o *hub* apenas recria o bit, aumenta a energia e o transmite para todas as outras interfaces. Sendo assim, Ethernet com uma topologia de estrela baseada em um *hub* também é uma LAN de difusão — sempre que um *hub* recebe um bit de uma de suas interfaces, ele envia uma cópia para todas as outras interfaces. Em particular, se um *hub* recebe quadros de duas diferentes interfaces ao mesmo tempo, ocorre uma colisão e os nós que criaram os quadros precisam retransmitir.

No começo dos anos 2000, Ethernet passou por outra grande mudança evolucionária. As instalações Ethernet continuaram a usar a topologia de estrela, mas o *hub* no núcleo foi substituído por um **comutador**. Examinaremos o comutador da Ethernet com mais atenção em outro ponto deste capítulo. Por enquanto, só mencionaremos que um comutador é não apenas "sem colisões", mas também um autêntico comutador de pacotes do tipo armazenar-e-repassar; mas, ao contrário dos roteadores, que operam até a camada 3, um comutador opera até a camada 2.

Estrutura do quadro Ethernet

Podemos aprender muito sobre a Ethernet examinando o quadro mostrado na Figura 5.20. Para colocar nossa discussão de quadros Ethernet em um contexto tangível, vamos considerar o envio de um datagrama IP de um hospedeiro a outro, estando os dois na mesma LAN Ethernet (por exemplo, a da Figura 5.17). (Embora a carga útil do nosso quadro Ethernet seja um diagrama IP, notamos que um quadro Ethernet também pode

carregar outros pacotes da camada de rede.) Seja o adaptador do remetente, adaptador A, com o endereço MAC AA-AA-AA-AA-AA-AA; seja o adaptador receptor, adaptador B, com o endereço MAC BB-BB-BB-BB-BB-BB. O adaptador remetente encapsula o datagrama IP dentro de um quadro Ethernet, o qual passa à camada física. O adaptador receptor recebe o quadro da camada física, extrai o datagrama IP e o passa para a de rede. Nesse contexto, vamos examinar os seis primeiros campos do quadro Ethernet, como ilustrados na Figura 5.20.

FIGURA 5.20 ESTRUTURA DO QUADRO ETHERNET

Preâmbulo	Endereço de destino	Endereço de origem	Tipo	Dados		CRC

- *Campo de dados (46 a 1.500 bytes)*. Esse campo carrega o datagrama IP. A unidade máxima de transferência (MTU) da Ethernet é 1.500 bytes. Isso significa que, se o datagrama IP exceder 1.500 bytes, o hospedeiro terá de fragmentar o datagrama, como discutimos na Seção 4.4.1. O tamanho mínimo do campo de dados é 46 bytes. Isso significa que, se um datagrama IP tiver menos do que 46 bytes, o campo de dados terá de ser "preenchido" de modo a completar os 46 bytes. Quando se usa o preenchimento, os dados passados à camada de rede contêm preenchimento, bem como um datagrama IP. A camada de rede usa o campo de comprimento do cabeçalho do datagrama IP para remover o preenchimento.
- *Endereço de destino (6 bytes)*. Esse campo contém o endereço MAC do adaptador de destino, BB-BB-BB-BB-BB-BB. Quando o adaptador B recebe um quadro Ethernet cujo endereço de destino é ou BB-BB-BB-BB-BB-BB, ou o endereço MAC de difusão, ele passa o conteúdo do campo de dados para a camada de rede. Se receber um quadro com qualquer outro endereço MAC, ele o descarta.
- *Endereço de origem (6 bytes)*. Esse campo contém o endereço MAC do adaptador que transmite o quadro para a LAN, neste exemplo, AA-AA-AA-AA-AA-AA.
- *Campo de tipo (2 bytes)*. O campo de tipo permite que a Ethernet multiplexe protocolos da camada de rede. Para entender isso, é preciso ter em mente que hospedeiros podem usar outros protocolos da camada de rede além do IP. Na verdade, um hospedeiro pode suportar vários protocolos da camada de rede e usar protocolos diferentes para aplicações diversas. Por essa razão, quando o quadro Ethernet chega ao adaptador B, o adaptador B precisa saber para qual protocolo da camada de rede ele deve passar (isto é, demultiplexar) o conteúdo do campo de dados. O IP e outros protocolos da camada de rede (por exemplo, Novell IPX ou AppleTalk) têm seu próprio número de tipo padronizado. Além disso, o protocolo ARP (discutido na seção anterior) tem seu próprio número de tipo, e se o quadro que chegar contiver um pacote ARP (por exemplo, tem um campo de tipo 0806 hexadecimal), o pacote ARP será demultiplexado até o protocolo ARP. Note que o campo de tipo é semelhante ao campo de protocolo no datagrama da camada de rede e aos campos de número de porta no segmento da camada de transporte; todos eles servem para ligar um protocolo de uma camada a um protocolo da camada acima.
- *Verificação de redundância cíclica (CRC) (4 bytes)*. Como discutido na Seção 5.2.3, a finalidade do campo de CRC é permitir que o adaptador receptor, o adaptador B, detecte se algum erro de bit foi introduzido no quadro.
- *Preâmbulo (8 bytes)*. O quadro Ethernet começa com um campo de preâmbulo de 8 bytes. Cada um dos primeiros 7 bytes do preâmbulo tem um valor de 10101010; o último byte é 10101011. Os primeiros 7 bytes do preâmbulo servem para "despertar" os adaptadores receptores e sincronizar seus relógios com o relógio do remetente. Por que os relógios poderiam estar fora de sincronia? Não esqueça que o adaptador A visa transmitir o quadro a 10 Mbits/s, 100 Mbits/s ou 1 Gbit/s, dependendo do tipo de LAN Ethernet. Contudo, como nada é absolutamente perfeito, o adaptador A não transmitirá o quadro exatamente à

mesma velocidade-alvo; sempre haverá alguma variação em relação a ela, uma variação que não é conhecida *a priori* pelos outros adaptadores na LAN. Um adaptador receptor pode sincronizar com o relógio do adaptador A apenas sincronizando os bits dos primeiros 7 bytes do preâmbulo. Os dois últimos bits do oitavo byte do preâmbulo (os primeiros dois 1s consecutivos) alertam o adaptador B de que "algo importante" está chegando.

Todas as tecnologias Ethernet fornecem serviço não orientado para conexão à camada de rede. Isto é, quando o adaptador A quer enviar um datagrama ao adaptador B, o adaptador A encapsula o datagrama em um quadro Ethernet e envia o quadro à LAN, sem se apresentar previamente a B. Esse serviço de camada 2 não orientado para conexão é semelhante ao serviço de datagrama de camada 3 do IP e ao serviço de camada 4 não orientado para conexão do UDP.

As tecnologias Ethernet fornecem um serviço não confiável à camada de rede. Especificamente, quando o adaptador B recebe um quadro do adaptador A, ele submete o quadro a uma verificação de CRC, mas não envia um reconhecimento quando um quadro passa na verificação de CRC nem um reconhecimento negativo quando o quadro não passa na verificação. Quando um quadro não passa na verificação de CRC, o adaptador B simplesmente o descarta. Assim, o adaptador A não tem a mínima ideia se o quadro que transmitiu passou na verificação de CRC. Essa falta de transporte confiável (na camada de enlace) ajuda a tornar a Ethernet simples e barata. Mas também significa que a sequência de datagramas passada à camada de rede pode ter lacunas.

Se houver lacunas por causa de quadros Ethernet descartados, a aplicação no hospedeiro B também verá essas lacunas? Como aprendemos no Capítulo 3, isso depende exclusivamente de a aplicação estar usando UDP ou TCP. Se estiver usando UDP, então a aplicação no hospedeiro B verá de fato lacunas nos dados. Por outro lado, se a aplicação estiver usando TCP, então o TCP no hospedeiro B não reconhecerá os dados contidos em quadros descartados, fazendo que o TCP no hospedeiro A retransmita. Note que, quando o TCP retransmite dados, estes eventualmente retornarão ao adaptador Ethernet no qual foram descartados. Assim, nesse sentido, a Ethernet

HISTÓRIA

Bob Metcalfe e a Ethernet

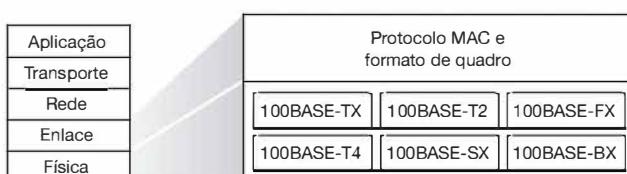
Quando era estudante de doutorado na Universidade de Harvard, no início da década de 1970, Bob Metcalfe trabalhava na ARPAnet no MIT. Durante seus estudos, ele tomou conhecimento do trabalho de Abramson com o ALOHA e os protocolos de acesso aleatório. Após ter concluído seu doutorado e pouco antes de começar um trabalho no PARC (Palo Alto Research Center) da Xerox, fez uma visita de três meses a Abramson e seus colegas da Universidade do Havaí, quando pôde observar, em primeira mão, a ALOHAnet. No PARC da Xerox, Metcalf conheceu os computadores Alto, que, em muitos aspectos, foram os predecessores dos equipamentos pessoais da década de 1980. Ele entendeu a necessidade de montar esses computadores em rede de um modo que não fosse dispendioso. Assim, munido com o que conhecia sobre ARPAnet, ALOHAnet e protocolos de acesso aleatório, Metcalfe, junto com seu colega David Boggs, inventou a Ethernet.

A Ethernet original de Metcalfe e Boggs executava a 2,94 Mbits/s e interligava até 256 hospedeiros a distâncias de até 1,5 km. Metcalfe e Boggs conseguiram que a maioria dos pesquisadores do PARC da Xerox se comunicasse por meio de seus computadores Alto. Então, Metcalfe forjou uma aliança entre a Xerox, a Digital e a Intel para estabelecer a Ethernet de 10 Mbits/s como padrão, ratificado pelo IEEE. A Xerox não demonstrou muito interesse em comercializar a Ethernet. Em 1979, Metcalfe abriu sua própria empresa, a 3Com, para desenvolver e comercializar tecnologia de rede, incluindo a tecnologia Ethernet. Em particular, a 3Com desenvolveu e comercializou placas Ethernet no início da década de 1980 para os então popularíssimos PCs da IBM. Metcalfe deixou a 3Com em 1990, quando a empresa tinha dois mil funcionários e 400 milhões de dólares de receita.

realmente retransmite dados, embora não saiba se está transmitindo um datagrama novo com dados novos, ou um datagrama que contém dados que já foram transmitidos pelo menos uma vez.

Tecnologias Ethernet

Em nossa discussão anterior, nos referimos à Ethernet como se fosse um único protocolo-padrão. Mas, na verdade, ela aparece em diferentes versões, com acrônimos um pouco confusos como 10BASE-T, 10BASE-2, 100BASE-T, 1000BASE-LX e 10GBASE-T. Essas e muitas outras tecnologias Ethernet foram padronizadas através dos anos pelos grupos de trabalho IEEE 802.3. Apesar de esses acrônimos parecerem confusos, existe uma ordem seguida. A primeira parte do acrônimo se refere à velocidade-padrão: 10, 100, 1.000 ou 10G, por 10 Megabits (por segundo), 100 Megabits e 10 Gigabits Ethernet, respectivamente. "BASE" se refere à banda-base, significando que a mídia física só suporta o tráfego da Ethernet; quase todos os padrões 802.3 são para banda-base. A parte final do acrônimo se refere à mídia física em si; a Ethernet é uma camada de enlace *e* uma camada física que inclui um cabo coaxial, fio de cobre e fibra. Em geral um "T" se refere a um cabo de par trançado de fios de cobre.

Historicamente, uma Ethernet era de início concebida como um segmento de um cabo coaxial. Os primeiros padrões 10BASE-2 e 10BASE-5 especificavam a Ethernet a 10 Mbits/s sobre dois tipos de cabos coaxiais, cada um limitado a um comprimento de 500 m. Extensões mais longas podiam ser obtidas usando um **repetidor** — um dispositivo da camada de enlace que recebe um sinal no lado de entrada, e regenera o sinal no lado de saída. Um cabo coaxial, como na Figura 5.20, corresponde muito bem a nossa visão da Ethernet como um meio de difusão — todos os quadros transmitidos por uma interface são recebidos em outras interfaces, e seu protocolo CDMA/CD resolve de maneira satisfatória o problema de acesso múltiplo. Os nós são apenas conectados ao cabo, e *voilà*, temos uma rede local!

A Ethernet passou por uma série de etapas de evolução ao longo dos anos, e a atual é muito diferente do projeto original da topologia de barramento que usava cabos coaxiais. Na maioria das instalações de hoje, os nós são conectados a um comutador via segmentos ponto a ponto feitos de cabos de pares trançados de fios de cobre ou cabos de fibra ótica, como demonstrado nas figuras 5.15-5.17.

No meio da década de 1990, a Ethernet foi padronizada em 100 Mbits/s, dez vezes mais rápida do que a de 10 Mbits/s. O formato de quadro e o protocolo Ethernet MAC original foram preservados, mas camadas físicas de alta velocidade foram definidas para fios de cobre (100BASE-T) e fibra (100BASE-FX, 100BASE-SX, 100BASE-BX). A Figura 5.21 mostra esses diferentes padrões e os formatos de quadro e protocolo Ethernet MAC comum. A Ethernet de 100 Mpbs é limitada a 100 m de distância por um cabo de par trançado, e vários quilômetros por fibra, o que permite a conexão de comutadores Ethernet em diferentes prédios.

A Gigabit Ethernet é uma extensão dos muito bem-sucedidos padrões 10 Mbits/s e 100 Mbits/s. Oferecendo uma velocidade bruta de 1.000 Mbits/s, mantém total compatibilidade com a imensa base instalada de equipamentos Ethernet. O padrão para a Gigabit Ethernet, formalmente conhecido como IEEE 802.3z, faz o seguinte:

- Usa o formato-padrão do quadro Ethernet (Figura 5.20) e é compatível com as tecnologias 10BASE-T e 100BASE-T. Isso permite fácil integração da Gigabit Ethernet com a base instalada de equipamentos Ethernet.

FIGURA 5.21 PADRÕES ETHERNET DE 100 Mbits/s: UMA CAMADA DE ENLACE COMUM, DIFERENTES CAMADAS FÍSICAS

Aplicação
Transporte
Rede
Enlace
Física
Protocolo MAC e formato de quadro
100BASE-TX
100BASE-T2
100BASE-FX
100BASE-T4
100BASE-SX
100BASE-BX

- Permite enlaces ponto a ponto, bem como canais de difusão compartilhados. Tais enlaces usam comutadores, ao passo que canais de difusão usam *hubs*, como descrito antes. No jargão da Gigabit Ethernet, os *hubs* são denominados *distribuidores com* buffer.
- Utiliza CSMA/CD para canais de difusão compartilhados. Para conseguir eficiência aceitável, a distância máxima entre os nós deve ser severamente limitada.
- Permite operação *full-duplex* a 1.000 Mbits/s em ambas as direções para canais ponto a ponto.

No início operando através de fibra ótica, a Gigabit Ethernet está disponível para ser instalada por meio de cabeamento UTP categoria 5. A Ethernet a 10 Gbits/s (10GBASE-T) foi padronizada em 2007, oferecendo uma capacidade de LAN Ethernet ainda mais alta.

Vamos concluir nossa discussão sobre a tecnologia Ethernet considerando uma dúvida que pode estar incomodando você. Na época da topologia de barramento e da topologia de estrela baseada em *hub*, a Ethernet era evidentemente um enlace de difusão (como definido na Seção 5.3), onde colisões de quadro ocorriam quando nós transmitiam ao mesmo tempo. Para lidar com essas colisões, o padrão Ethernet incluiu o protocolo CSMA/CD, que é de particular eficácia para transmissões de LAN abrangendo uma pequena região geográfica. Mas se o uso atual prevalente da Ethernet é baseado em comutadores com a topologia de estrela, usando o modo de comutação armazenar-e-repassar, existe mesmo a necessidade de se usar um protocolo Ethernet MAC? Como veremos em breve, um comutador coordena suas transmissões e nunca repassa mais de um quadro por vez na mesma interface. Além disso, comutadores modernos são *full-duplex*, de modo que um comutador e um nó possam enviar quadros um ao outro ao mesmo tempo sem interferência. Em outras palavras, em uma LAN Ethernet baseada em comutador, não há colisões e, portanto, não existe a necessidade de um protocolo MAC!

Como vimos, a Ethernet atual é *muito* diferente da original concebida por Metcalfe e Boggs há mais de 30 anos — as velocidades tiveram um aumento de três ordens de grandeza, os quadros Ethernet são transportados por uma variedade de mídias, as Ethernets comutadas se tornaram dominantes e até mesmo o protocolo MAC é muitas vezes desnecessário! Será que tudo isso *realmente* ainda é Ethernet? A resposta é, obviamente, "sim, por definição". No entanto, é interessante observar que, por todas essas mudanças, existiu uma constante que continuou inalterada por mais de 30 anos — o formato do quadro Ethernet. Talvez esta seja a única peça central verdadeira e eterna do padrão Ethernet.

5.4.3 Comutadores da camada de enlace

Até aqui, temos sido deliberadamente vagos sobre como um comutador* trabalha e o que ele faz. A função de um comutador é receber quadros da camada de enlace e repassá-los para enlaces de saída; estudaremos essa função de repasse detalhadamente em breve. O comutador em si é **transparente** aos hospedeiros e roteadores na sub-rede; ou seja, um nó endereça um quadro a outro nó (em vez de endereçar o quadro ao comutador) que alegremente envia o quadro à LAN, sem saber que um comutador receberá o quadro e o repassará. A velocidade com que os quadros chegam a qualquer interface de saída do comutador pode temporariamente exceder a capacidade do enlace daquela interface. Para resolver esse problema, interfaces de saídas do comutador têm buffers, da mesma forma que uma interface de saída de um roteador tem buffers para datagramas. Vamos agora observar mais atentamente o funcionamento de um comutador.

Repasse e filtragem

Filtragem é a capacidade de um comutador que determina se um quadro deve ser repassado para alguma interface ou se deve apenas ser descartado. **Repasse** é a capacidade de um comutador que determina as interfaces para as quais um quadro deve ser dirigido e então dirigir o quadro a essas interfaces. Filtragem e repasse por

* Embora o termo "switch" seja jargão corrente entre os profissionais de rede preferimos traduzi-lo por "comutador", já que "switch" pode ser empregado em diferentes sentidos e não foi padronizado pelo IEEE (N. R.)

comutadores são feitos com uma **tabela de comutação**. A tabela de comutação contém registros para alguns hospedeiros e roteadores da LAN, mas não necessariamente para todos. Um registro na tabela de comutação contém (1) o endereço MAC, (2) a interface do comutador que leva em direção a esse endereço MAC e (3) o horário em que o registro foi colocado na tabela. Um exemplo de tabela de comutação para a LAN da Figura 5.15 é mostrado na Figura 5.22. Embora essa descrição de repasse de quadros possa parecer semelhante à nossa discussão de repasse de datagramas no Capítulo 4, logo veremos que há diferenças significativas. Uma diferença importante é que os comutadores repassam pacotes baseados em endereços MAC, em vez de endereços IP. Também veremos que uma tabela de comutação é montada de maneira diferente da tabela de roteamento de um roteador.

FIGURA 5.22 PARTE DE UMA TABELA DE COMUTAÇÃO PARA O COMUTADOR MAIS ACIMA NA FIGURA 5.15

Endereço	Interface	Horário
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
....	....	....

Para entender como funcionam a filtragem e o repasse por comutadores, suponha que um quadro com endereço de destino DD-DD-DD-DD-DD-DD chegue ao comutador na interface *x*. O comutador indexa sua tabela com o endereço MAC DD-DD-DD-DD-DD-DD. Há três casos possíveis:

- Não existe entrada na tabela para DD-DD-DD-DD-DD-DD. Nesse caso, o comutador repassa cópias do quadro para os buffers de saída que precedem *todas* as interfaces, exceto a interface *x*. Em outras palavras, se não existe entrada para o endereço de destino, o comutador transmite o quadro por difusão.
- Existe uma entrada na tabela, associando DD-DD-DD-DD-DD-DD com a interface *x*. Nesse caso, o quadro está vindo de um segmento da LAN que contém o adaptador DD-DD-DD-DD-DD-DD. Não havendo necessidade de repassar o quadro para qualquer outra interface, o comutador realiza a função de filtragem ao descartar o quadro.
- Existe uma entrada na tabela, associando DD-DD-DD-DD-DD-DD com a interface $y \neq x$. Nesse caso, o quadro precisa ser repassado ao segmento da LAN conectado à interface *y*. O comutador realiza sua função de repasse ao colocar o quadro em um buffer de saída que precede a interface *y*.

Vamos examinar essas regras para a rede da Figura 5.15 e sua tabela de comutação na Figura 5.22 Suponha que um quadro com endereço de destino 62-FE-F7-11-89-A3 chegue ao comutador vindo da interface 1. O comutador examina sua tabela e vê que o destino está no segmento de LAN conectado à interface 1 (isto é, do departamento de engenharia elétrica). Isso significa que o quadro já foi transmitido por difusão no segmento de LAN que contém o destino. Por conseguinte, o comutador filtra (isto é, descarta) o quadro. Agora suponha que um quadro com o mesmo endereço de destino chegue da interface 2. O comutador novamente examina sua tabela e verifica que o destino está na direção da interface 1; por conseguinte, ele repassa o quadro para o buffer de saída que precede a interface 1. Com este exemplo, deve ficar claro que, enquanto a tabela de comutação permanecer completa e precisa, o comutador encaminha quadros até seus destinos sem qualquer transmissão por difusão.

Assim, nesse sentido, o comutador é "mais esperto" do que um *hub*. Mas como uma tabela de comutação é configurada afinal? Existem equivalentes de camadas de enlace a protocolos de roteamento da camada de rede? Ou um gerente sobrecarregado de serviço deve configurar manualmente a tabela de comutação?

Autoaprendizagem

Um comutador tem a maravilhosa propriedade (em especial, para o administrador de rede, que quase sempre está sobrecarregado) de montar sua tabela de modo automático, dinâmico e autônomo — sem nenhuma intervenção de um administrador de rede ou de um protocolo de configuração. Em outras palavras, comutadores são **autodidatas**. Essa capacidade é obtida da seguinte forma:

1. A tabela de comutação inicialmente está vazia.
2. Para cada quadro recebido em uma interface, o comutador armazena em sua tabela (1) o endereço MAC que está no *campo de endereço de origem* do quadro, (2) a interface da qual veio o quadro e (3) o horário corrente. Dessa maneira, o comutador registra em sua tabela o segmento da LAN no qual reside o nó remetente. Se cada hospedeiro na LAN mais cedo ou mais tarde enviar um quadro, então cada um deles por fim estará registrado na tabela.
3. O comutador apagará um endereço na tabela se nenhum quadro que tenha aquele endereço como endereço de origem for recebido após certo período de tempo (o **tempo de envelhecimento**). Desse modo, se um PC for substituído por outro (com um adaptador diferente), o endereço MAC do PC original acabará sendo expurgado da tabela de comutação.

Vamos examinar a propriedade de aprendizagem automática para a rede da Figura 5.15 e sua tabela de comutação correspondente, apresentada na Figura 5.22. Suponha que, no horário 9h39min, um quadro com endereço de origem 01-12-23-34-45-56 venha da interface 2. Suponha também que esse endereço não esteja na tabela de comutação. Então, o comutador anexa um novo registro à tabela, conforme mostra a Figura 5.23.

Continuando com esse mesmo exemplo, suponha ainda que o tempo de envelhecimento para esse comutador seja 60 minutos e que nenhum quadro com endereço de origem 62-FE-F7-11-89-A3 chegue ao comutador entre 9h32min e 10h32min. Então, no horário 10h32min, o comutador remove esse endereço de sua tabela.

Comutadores são **dispositivos do tipo *plug-and-play*** porque não requerem a intervenção de um administrador ou de um usuário da rede. Um administrador de rede que quiser instalar um comutador não precisa fazer nada mais do que conectar os segmentos de LAN às interfaces do comutador. O administrador não precisa configurar as tabelas de comutação na hora da instalação nem quando um hospedeiro é removido de um dos segmentos de LAN. Comutadores também são *full-duplex*, ou seja, qualquer interface do comutador pode enviar e receber ao mesmo tempo.

FIGURA 5.23 O COMUTADOR APRENDE A LOCALIZAÇÃO DO ADAPTADOR COM ENDEREÇO 01-12-23-34-45-56

Endereço	Interface	Horário
01-12-23-34-45-56	2	9:39
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
....	....	....

Propriedades de comutação da camada de enlace

Tendo descrito as operações básicas da comutação da camada de enlace, vamos considerar suas propriedades e funcionalidades. Podemos identificar diversas vantagens no uso de comutadores, em vez de se usarem enlaces de difusão como barramentos ou topologias de estrela baseadas em *hub*:

- *Eliminação de colisões*: Em uma LAN montada com comutadores (e sem *hubs*), não existe desperdício de banda causado por colisões! Os comutadores armazenam os quadros e nunca transmitem mais de um quadro em um segmento ao mesmo tempo. Como em um roteador, a vazão máxima agregada de um comutador é a soma da velocidade de todas as interfaces do comutador. Portanto, os comutadores oferecem uma melhoria de desempenho significativa em relação às LANs com enlaces de difusão.
- *Enlaces heterogêneos*: Uma vez que o comutador isola um enlace do outro, os diferentes enlaces na LAN conseguem operar em diferentes velocidades e podem ser executados por diferentes mídias. Por exemplo, o comutador mais acima na Figura 5.22 poderia ter três enlaces de cobre 1000BASE-T de

SEGURANÇA EM FOCO

Analisando uma LAN comutada: envenenamento de comutador

Quando um hospedeiro é conectado a um comutador, em geral só recebe quadros que estão sendo enviados explicitamente a ele. Por exemplo, considere uma LAN comutada na Figura 5.17. Quando o hospedeiro A envia um quadro ao hospedeiro B, e há um registro para B na tabela de comutação, então o comutador repassa o quadro *somente* para B. Se o hospedeiro C estiver executando um analisador de quadros, o hospedeiro C não poderá analisar esse quadro de A-para-B. Assim, em um ambiente de LAN comutada (ao contrário de um ambiente de enlace de difusão, como LANs 802.11 ou LANs Ethernet baseadas em *hub*), é mais difícil que um invasor analise quadros. *Porém*, como o comutador envia por difusão quadros que possuem endereços de destino que não estão na tabela de comutação, o invasor em C ainda poderá sondar alguns quadros que não são endereçados de modo explícito a C. Além disso, um farejador poderá vasculhar todos os quadros de difusão Ethernet com endereço de destino de difusão FF–FF–FF–FF–FF–FF. Um ataque bem conhecido contra um comutador, denominado **envenenamento de comutador**, é enviar uma grande quantidade de pacotes ao comutador com muitos endereços MAC de origem falsos e diferentes, enchendo assim a tabela de comutação com registros falsos e não deixando espaço para os endereços MAC dos hospedeiros legítimos. Isso faz o comutador enviar a maioria dos quadros por difusão, podendo então ser apanhados pelo esquadrinhador [Skoudis, 2006]. Visto que esse ataque é bem complexo, mesmo para um invasor sofisticado, os comutadores são muito menos vulneráveis à análise do que os *hubs* e as LANs sem fio.

1 Gbit/s, dois enlaces de fibra 100BASE-FX de 100 Mbits/s e um enlace de cobre 100BASE-T. Assim, um comutador é ideal para misturar equipamento legado e novo.

- *Gerenciamento.* Além de oferecer mais segurança (ver a nota "Foco na segurança"), um comutador também facilita o gerenciamento da rede. Por exemplo, se um adaptador apresenta defeito e envia continuamente quadros Ethernet (chamado adaptador tagarela), um comutador pode detectar o problema e desconectar internamente o adaptador com defeito. Com esse recurso, o administrador da rede não precisa se levantar de madrugada e dirigir até o trabalho para corrigir o problema. De modo semelhante, um cabo cortado desconecta apenas o hospedeiro que o estava usando para conectar o comutador. Nos dias do cabo coaxial, muitos gerentes de rede gastavam horas "percorrendo as linhas" (ou, mais precisamente, "arrastando-se pelo chão") para achar a interrupção que paralisou a rede inteira. Conforme discutimos no Capítulo 9 (Gerenciamento de rede), os comutadores também colhem estatísticas sobre uso da largura de banda, taxas de colisão e tipos de tráfego, e tornam essa informação disponível para o gerente da rede. Tal informação pode ser usada para depurar e corrigir problemas, além de planejar como a LAN deverá evoluir no futuro. Os pesquisadores estão explorando a inclusão de ainda mais funcionalidade de gerenciamento para as LANs Ethernet nos protótipos implementados [Casado, 2007; Koponen, 2011].

Comutadores *versus* roteadores

Como aprendemos no Capítulo 4, roteadores são comutadores de pacotes do tipo armazena-e-repassa, que transmitem pacotes usando endereços da camada de rede. Embora um comutador também seja um comutador de pacotes do tipo armazena-e-repassa, ele é em essência diferente de um roteador, pois repassa pacotes usando endereços MAC. Enquanto um roteador é um comutador de pacotes da camada 3, um comutador opera com protocolos da camada 2.

Mesmo sendo fundamentalmente diferentes, é comum que os administradores de rede tenham de optar entre um comutador e um roteador ao instalar um dispositivo de interconexão. Por exemplo, para a rede da Figura 5.15,

o administrador de rede poderia com facilidade ter optado por usar um roteador, em vez de um comutador, para conectar as LANs de departamento, servidores e roteador de borda da Internet. Na verdade, um roteador permitiria a comunicação interdepartamental sem criar colisões. Dado que ambos, comutadores e roteadores, são candidatos a dispositivos de interconexão, quais são os prós e os contras das duas técnicas?

Vamos considerar, inicialmente, os prós e os contras de comutadores. Como já dissemos, comutadores são do tipo *plug-and-play*, uma propriedade que é apreciada por todos os administradores de rede atarefados do mundo. Eles também podem ter velocidades relativamente altas de filtragem e repasse — como ilustra a Figura 5.24, têm de processar quadros apenas até a camada 2, enquanto roteadores têm de processar pacotes até a camada 3. Por outro lado, para evitar a circulação dos quadros por difusão, a topologia de uma rede de comutação está restrita a uma *spanning tree*. E mais, uma rede de comutação de grande porte exigiria, nos hospedeiros e roteadores, tabelas ARP também grandes, gerando tráfego e processamento ARP substanciais. Além do mais, comutadores são suscetíveis a tempestades de difusão — se um hospedeiro se desorganiza e transmite uma corrente sem fim de quadros Ethernet por difusão, os comutadores repassam todos esses quadros, causando o colapso da rede inteira.

Agora, vamos considerar os prós e os contras dos roteadores. Como na rede o endereçamento muitas vezes é hierárquico (e não linear, como o MAC), os pacotes em geral não ficam circulando nos roteadores, mesmo quando a rede tem trajetos redundantes. (Na verdade, eles podem circular quando as tabelas de roteadores estão mal configuradas; mas, como aprendemos no Capítulo 4, o IP usa um campo de cabeçalho de datagrama especial para limitar a circulação.) Assim, pacotes não ficam restritos a uma topologia de *spanning tree* e podem usar o melhor trajeto entre origem e destino. Como roteadores não sofrem essa limitação, eles permitiram que a Internet fosse montada com uma topologia rica que inclui, por exemplo, múltiplos enlaces ativos entre a Europa e a América do Norte. Outra característica dos roteadores é que eles fornecem proteção de *firewall* contra as tempestades de difusão da camada 2. Talvez sua desvantagem mais significativa seja o fato de não serem do tipo *plug-and-play* — eles e os hospedeiros que a eles se conectam precisam de seus endereços IP para ser configurados. Além disso, roteadores muitas vezes apresentam tempo de processamento por pacote maior do que comutadores, pois têm de processar até os campos da camada 3.

Dado que comutadores e roteadores têm seus prós e contras (como resumido na Tabela 5.1), quando uma rede institucional (por exemplo, de um *campus* universitário ou uma rede corporativa) deveria usar comutadores e quando deveria usar roteadores? Em geral, redes pequenas, com algumas centenas de hospedeiros, têm uns poucos segmentos de LAN. Para essas, comutadores serão satisfatórios, pois localizam o tráfego e aumentam a vazão agregada sem exigir nenhuma configuração de endereços IP. Mas redes maiores, com milhares de hospedeiros, em geral incluem roteadores (além de comutadores). Roteadores fornecem isolamento de tráfego mais robusto, controlam tempestades de difusão e usam rotas "mais inteligentes" entre os hospedeiros da rede.

Para mais informações sobre os prós e os contras das redes comutadas e roteadas, bem como sobre como a tecnologia de LAN comutada pode ser estendida para acomodar duas ordens de magnitude de hospedeiros a mais que a Ethernet atual, consulte Meyers [2004]; Kim [2008].

FIGURA 5.24 PROCESSAMENTO DE PACOTES EM COMUTADORES, ROTEADORES E HOSPEDEIROS

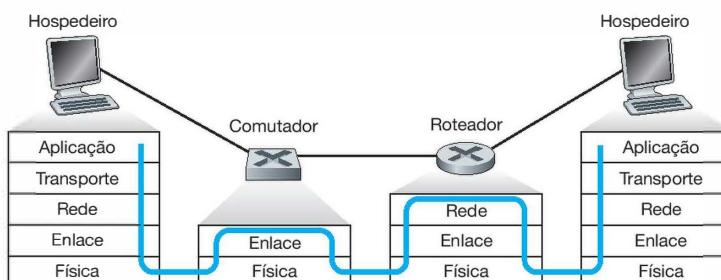
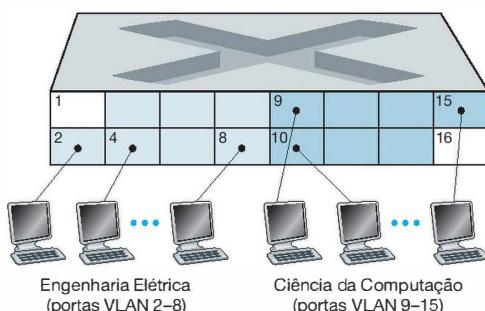



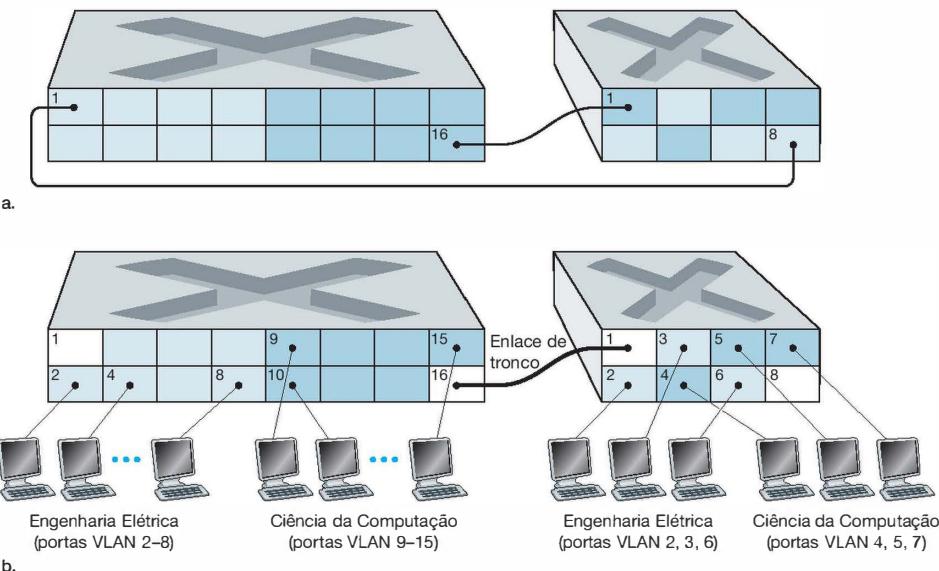
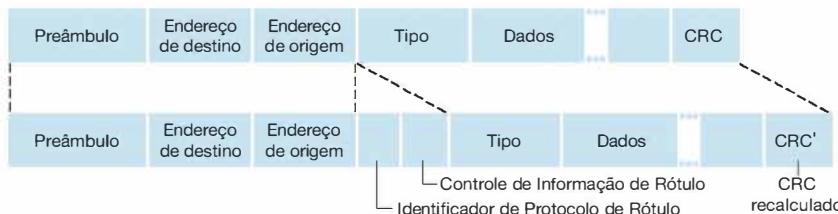
TABELA 5.1 COMPARAÇÃO ENTRE AS CARACTERÍSTICAS TÍPICAS DE DISPOSITIVOS DE INTERCONEXÃO POPULARES

	Hubs	Roteadores	Comutadores
Isolamento de tráfego	Não	Sim	Sim
Plug-and-play	Sim	Não	Sim
Roteamento ideal	Não	Sim	Não

5.4.4 Redes locais virtuais (VLANs)

Na discussão anterior sobre a Figura 5.15, notamos que as LANs institucionais modernas com frequência são configuradas hierarquicamente, cada grupo de trabalho (departamento) tendo seu próprio comutador de LAN conectado ao comutador de LAN de outros grupos via uma hierarquia de comutadores. Embora tal configuração funcione bem em um mundo ideal, o mundo real é bem diferente. Três desvantagens podem ser identificadas na configuração da Figura 5.15:

- *Falta de isolamento do tráfego*. Apesar de a hierarquia localizar o tráfego de grupos dentro de um único comutador, o tráfego de difusão (por exemplo, quadros carregando mensagens ARP e DHCP ou quadros com endereços de destino que ainda não foram descobertos por um comutador com a autoaprendizagem) tem que ainda percorrer toda a rede institucional. Limitar o escopo desse tráfego de difusão aprimoraria o desempenho da LAN. Talvez mais importante que isso, também seria desejável limitar esse tráfego por razões de segurança e privacidade. Por exemplo, se um grupo contém a equipe da gerência executiva de uma empresa e outro grupo contém funcionários decepcionados executando o analisador de pacotes Wireshark, o gerente de rede talvez prefira que o tráfego da gerência executiva não alcance os computadores dos funcionários. Esse tipo de isolamento pode ser substituído trocando o comutador central da Figura 5.15 por um roteador. Em breve veremos que esse isolamento também pode ser realizado por meio de uma solução de comutação (camada 2).
- *Uso ineficiente de comutadores*. Se, em vez de três, a instituição tivesse dez grupos, os dez comutadores de primeiro nível seriam necessários. Se cada grupo fosse pequeno, digamos, com menos de dez pessoas, um comutador de 96 portas seria suficiente para atender a todos, mas esse único comutador não fornece isolamento de tráfego.
- *Gerenciamento de usuários*. Se um funcionário se locomove entre os grupos, o cabeamento físico deve ser mudado para conectá-lo a um comutador diferente na Figura 5.15. Funcionários pertencentes a dois grupos dificultam o problema.

Felizmente, cada uma dessas dificuldades pode ser resolvida com um comutador que suporte **Redes Locais Virtuais (VLANs)**. Como o nome já sugere, um comutador que suporta VLANs permite que diversas redes locais virtuais sejam executadas por meio de uma única infraestrutura *física* de uma rede local *virtual*. Hospedeiros dentro de uma VLAN se comunicam como se eles (e não outros hospedeiros) estivessem conectados ao comutador. Em uma VLAN baseada em portas, as portas (interfaces) do comutador são divididas em grupos pelo gerente da rede. Cada grupo constitui uma VLAN, com as portas em cada VLAN formando um domínio de difusão (por exemplo, o tráfego de difusão de uma porta só pode alcançar outras portas no grupo). A Figura 5.25 mostra um único comutador com 16 portas. As interfaces de 2 a 8 pertencem à VLAN EE, enquanto as de 9 a 15 pertencem à VLAN CS (portas de 1 e 16 não são atribuídas). Essa VLAN soluciona todas as dificuldades citadas — quadros de VLAN EE e CS são isolados uns dos outros, os dois comutadores na Figura 5.15 foram substituídos por um único comutador, e se o usuário da porta de comutador 8 se juntar ao Departamento CS, o operador da rede apenas reconfigura o software da VLAN para que a porta 8 seja associada com a VLAN CS. Qualquer um poderia imaginar facilmente como o comutador VLAN opera e é configurado — o gerente de rede declara uma interface pertencente a uma dada VLAN (com portas não declaradas pertencentes à VLAN padrão) usando um

FIGURA 5.25 COMUTADOR ÚNICO COM DUAS VLANs CONFIGURADAS

software de gerenciamento de comutadores, uma tabela de mapeamento porta-VLAN é mantida dentro do comutador; e um hardware de comutação somente entrega quadros entre as portas pertencentes à mesma VLAN.

Mas, isolando completamente as duas VLANs, criamos um novo problema! Como o tráfego do departamento EE pode ser enviado para o departamento CS? Uma maneira de se lidar com isso seria conectar uma porta de comutação da VLAN (exemplo, porta 1 na Figura 5.25) a um roteador externo, configurando aquela porta para que pertença às VLANs de ambos os departamentos EE e CS. Nesse caso, apesar de eles compartilharem um mesmo roteador físico, a configuração faria parecer que têm comutadores diferentes conectados por um roteador. Um datagrama IP, indo do departamento EE para o CS, passaria primeiro pela VLAN EE para alcançar o roteador e depois seria encaminhado de volta pelo roteador através da VLAN CS até o hospedeiro CS. Felizmente, os fornecedores de comutadores facilitam as configurações para o gerente de rede. Eles montam um dispositivo único que contém um comutador VLAN *e* um roteador, para que um roteador externo não seja necessário. Uma lição de casa ao final do capítulo explora esse exemplo mais em detalhes.

Voltando à Figura 5.15, vamos supor que, em vez de termos um departamento separado de Engenharia da Computação, parte do corpo docente de EE e de CS esteja alojada em um prédio separado, onde (é claro!) eles precisariam de acesso à rede, e (é claro!) gostariam de fazer parte do VLAN de seu departamento. A Figura 5.26 mostra um segundo comutador com 8 entradas, onde as entradas do comutador foram definidas como pertencentes à VLAN EE ou CS, conforme necessário. Mas como esses dois comutadores seriam interconectados? Uma solução fácil seria definir uma entrada como pertencente à VLAN CS em cada comutador (e da mesma forma para a VLAN EE) e conectá-las umas às outras, como demonstrado na Figura 5.26(a). No entanto, essa solução não permite crescimento, já que *N* VLANs exigiriam *N* portas em cada comutador para simplesmente interconectar os dois comutadores.

Uma abordagem mais escalável para interconectar os comutadores das VLANs é conhecido como **entroncamento de VLANs**. Na técnica de entrocamento de VLANs, mostrada na Figura 5.26(b), uma porta especial em cada comutador (porta 16 no comutador esquerdo e porta 1 no direito) é configurada como uma porta de tronco para interconectar os dois comutadores da VLAN. A porta de tronco pertence a todas as VLANs, e quadros enviados a qualquer VLAN são encaminhados pelo enlace de tronco ao outro comutador. Mas isso gera mais uma dúvida: como um comutador "sabe" que um quadro que está chegando a uma porta de tronco pertence a uma VLAN específica? O IEEE definiu um formato de quadro estendido, 802.1Q, para quadros atravessando o tronco VLAN. Conforme mostrado na Figura 5.27, o quadro 802.1Q consiste no quadro padrão Ethernet com um **rótulo de VLAN** de quatro bytes adicionado no cabeçalho que transporta a identidade da VLAN à qual o quadro pertence. O rótulo da VLAN é adicionado ao quadro pelo comutador no lado de envio do tronco de VLAN, analisado, e removido pelo comutador no lado de recebimento do tronco. O próprio rótulo da VLAN consiste em um campo de 2 bytes chamado Rótulo de Identificação de Protocolo (*Tag Protocol Identifier* — TPID) (com um valor hexadecimal fixo de 81-00), um campo de 2 bytes de Controle de Informação de Rótulo contendo um campo de identificação de VLAN com 12 bits, e um campo de prioridade com 3 bits semelhante em propósito ao campo TOS do datagrama IP.

FIGURA 5.26 CONECTANDO 2 COMUTADORES DA VLAN A DUAS VLANS: (A) 2 CABOS (B) ENTRONCADOS

FIGURA 5.27 QUADRO ETHERNET ORIGINAL (NO ALTO); QUADRO VLAN ETHERNET 802.1Q-TAGGED (EMBAIXO)

Nesta análise, só fizemos uma breve citação sobre VLANs e focamos em VLANs baseadas em portas. Deveríamos mencionar também que as VLANs podem ser definidas de diversas maneiras. Em uma VLAN baseada em MAC, o administrador de rede especifica o grupo de endereços MAC que pertence a cada VLAN; quando um dispositivo é conectado a uma porta, esta é conectada à VLAN apropriada com base no endereço MAC do dispositivo. As VLANs também podem ser definidas por protocolos da camada de rede (por exemplo, IPv4, IPv6 ou Appletalk) e outros critérios. Veja o padrão 802.1Q [IEEE802.1q, 2005] para obter mais informações.

5.5 VIRTUALIZAÇÃO DE ENLACE: UMA REDE COMO CAMADA DE ENLACE

Como este capítulo trata de protocolos da camada de enlace, e dado que estamos chegando ao fim, vamos refletir um pouco sobre como evoluiu o que entendemos como *enlace*. Começamos o capítulo considerando que

um enlace é um fio físico que conecta dois hospedeiros comunicantes. Quando estudamos protocolos de acesso múltiplo, vimos que vários hospedeiros podiam ser conectados por um fio compartilhado e que o "fio" que conectava os hospedeiros podia ser o espectro de rádio ou qualquer outro meio. Isso nos levou a ver o enlace, de modo um pouco mais abstrato, como um canal, em vez de um fio. Quando estudamos LANs Ethernet (Figura 5.15) vimos que, na verdade, os meios de interconexão poderiam ser uma infraestrutura de comutação bastante complexa. Durante toda essa evolução, entretanto, os hospedeiros sempre mantiveram a visão do meio de conexão apenas como um canal da camada de enlace conectando dois ou mais hospedeiros. Vimos, por exemplo, que um hospedeiro Ethernet pode facilmente ficar inconsciente do fato de estar ligado a outros hospedeiros de LAN por um único segmento curto de LAN (Figura 5.17) ou por uma LAN comutada geograficamente dispersa (Figura 5.15) ou pela VLAN (Figura 5.26).

No caso de uma conexão com modem discado entre dois hospedeiros, o enlace que conecta os dois é, na verdade, a rede de telefonia — uma rede global de telecomunicações logicamente separada, com seus próprios comutadores, enlaces e pilhas de protocolos para transferência e sinalização de dados. Entretanto, do ponto de vista da camada de rede da Internet, a conexão discada por meio da rede de telefonia é vista como um simples "fio". Nesse sentido, a Internet virtualiza a rede de telefonia, considerando-a uma tecnologia da camada de enlace que provê conectividade da camada de enlace entre dois hospedeiros da Internet. Lembre-se de que, quando discutimos redes de sobreposição no Capítulo 2, dissemos que, de modo semelhante, essa rede vê a Internet como um meio de prover conectividade entre nós sobrepostos, procurando sobrepor-se à Internet do mesmo modo que a Internet se sobrepõe à rede de telefonia.

Nesta seção consideraremos redes MPLS (*Multiprotocol Label Switching*). Diferentemente da rede de telefonia de comutação de circuitos, as redes MPLS, são, de direito, redes de comutação de pacotes por circuitos virtuais. Elas têm seus próprios formatos de pacotes e comportamentos de repasse. Assim, de um ponto de vista pedagógico, é bem coerente discutir MPLS quando estudamos a camada de rede ou a de enlace. Todavia, do ponto de vista da Internet, podemos considerar a MPLS, assim como a rede de telefonia e a Ethernet comutada, tecnologias da camada de enlace que servem para interconectar dispositivos IP. Assim, consideremos as redes MPLS ao discutirmos a camada de enlace. Redes *frame-relay* e ATM também podem ser usadas para interconectar dispositivos IP, embora representem uma tecnologia ligeiramente mais antiga (mas ainda disponibilizada), que não será discutida aqui; se quiser saber mais detalhes consulte Goralski [1999], um livro de fácil leitura. Nosso estudo de MPLS terá de ser breve, pois livros inteiros podem ser escritos (e foram) sobre essas redes. Recomendamos Davie [2000] para obter detalhes sobre MPLS. Aqui, focalizaremos principalmente como tais redes servem para interconectar dispositivos IP, embora as tecnologias subjacentes também sejam examinadas com um pouco mais de profundidade.

5.5.1 Comutação de Rótulos Multiprotocolo (MPLS)

A Comutação de Rótulos Multiprotocolo (MPLS — *Multiprotocol Label Switching*) evoluiu dos inúmeros esforços realizados pela indústria de meados ao final da década de 1990 para melhorar a velocidade de repasse de roteadores IP, adotando um conceito fundamental do mundo das redes de circuitos virtuais: um rótulo de tamanho fixo. O objetivo não era abandonar a infraestrutura de repasse de datagramas IP com base no destino em favor de rótulos de tamanho fixo e circuitos virtuais, mas aumentá-la rotulando datagramas seletivamente e permitindo que roteadores repassassem datagramas com base em rótulos de tamanho fixo (em vez de endereços de destino IP), quando possível. O importante é que essas técnicas trabalhavam de mãos dadas com o IP, usando endereçamento e roteamento IP. A IETF reuniu esses esforços no protocolo MPLS [RFC 3031, RFC 3032] misturando de fato técnicas de circuitos virtuais em uma rede de datagramas com roteadores.

Vamos começar nosso estudo do MPLS considerando o formato de um quadro da camada de enlace que é manipulado por um roteador habilitado para MPLS. A Figura 5.28 mostra que um quadro da camada de enlace transmitido entre dispositivos habilitados para MPLS tem um pequeno cabeçalho MPLS adicionado entre o cabeçalho de camada 2 (por exemplo, Ethernet) e o cabeçalho de camada 3 (isto é, IP). O RFC 3032 define o

formato do cabeçalho MPLS para esses enlaces; cabeçalhos de redes ATM e de frame relay também são definidos em outros RFCs. Entre os campos no cabeçalho MPLS estão o rótulo (que desempenha o papel de identificador de circuito virtual que estudamos na Seção 4.2.1), 3 bits reservados para uso experimental, um único bit, S, que é usado para indicar o final de uma série de rótulos MPLS "empilhados" (um tópico avançado que não será abordado aqui) e um campo de tempo de vida.

A Figura 5.28 deixa logo evidente que um quadro melhorado com MPLS só pode ser enviado entre roteadores habilitados para MPLS (já que um roteador não habilitado para MPLS ficaria bastante confuso ao encontrar um cabeçalho MPLS onde esperava encontrar o IP!). Um roteador habilitado para MPLS é em geral denominado **roteador de comutação de rótulos**, pois repassa um quadro MPLS consultando o rótulo MPLS em sua tabela de repasse e, então, passando imediatamente o datagrama para a interface de saída apropriada. Assim, o roteador habilitado para MPLS *não* precisa extrair o endereço de destino e executar uma busca para fazer a compatibilização com o prefixo mais longo na tabela de repasse. Mas como um roteador sabe se seu vizinho é realmente habilitado para MPLS e como sabe qual rótulo associar com determinado destino IP? Para responder a essas perguntas, precisaremos estudar a interação entre um grupo de roteadores habilitados para MPLS.

No exemplo da Figura 5.29, os roteadores R1 a R4 são habilitados para MPLS. R5 e R6 são roteadores IP padrão. R1 anunciou a R2 e R3 que ele (R1) pode rotear para o destino A, e que um quadro recebido com rótulo MPLS 6 será repassado ao destino A. O roteador R3 anunciou ao roteador R4 que ele (R4) pode rotear para os destinos A e D e que os quadros que estão chegando e que portam os rótulos MPLS 10 e 12 serão comutados na direção desses destinos. O roteador R2 também anunciou ao roteador R4 que ele (R2) pode alcançar o destino A e que um quadro recebido portando o rótulo MPLS 8 será comutado na direção de A. Note que o roteador R4 agora

FIGURA 5.28 CABEÇALHO MPLS: LOCALIZADO ENTRE OS CABEÇALHOS DA CAMADA DE ENLACE E DA CAMADA DE REDE

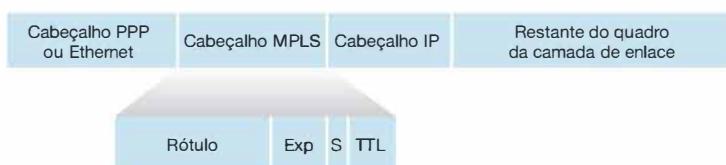
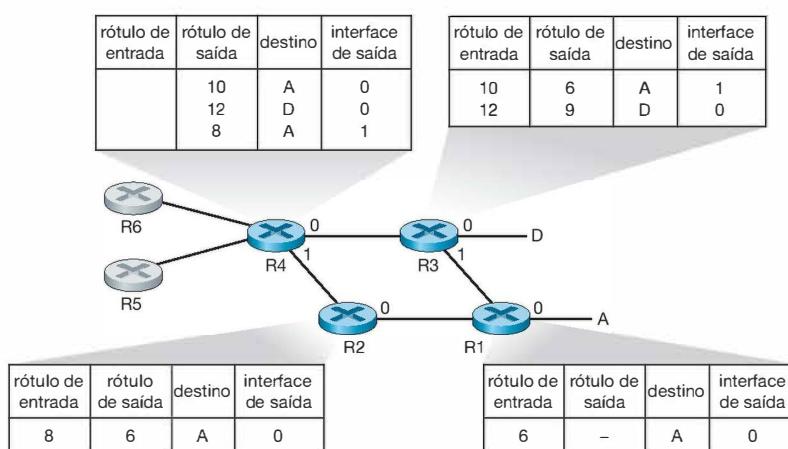



FIGURA 5.29 REPASSE MELHORADO COM MPLS

rótulo de entrada	rótulo de saída	destino	interface de saída
	10	A	0
	12	D	0
	8	A	1

rótulo de entrada	rótulo de saída	destino	interface de saída
10	6	A	1
12	9	D	0

rótulo de entrada	rótulo de saída	destino	interface de saída
8	6	A	0

rótulo de entrada	rótulo de saída	destino	interface de saída
6	–	A	0

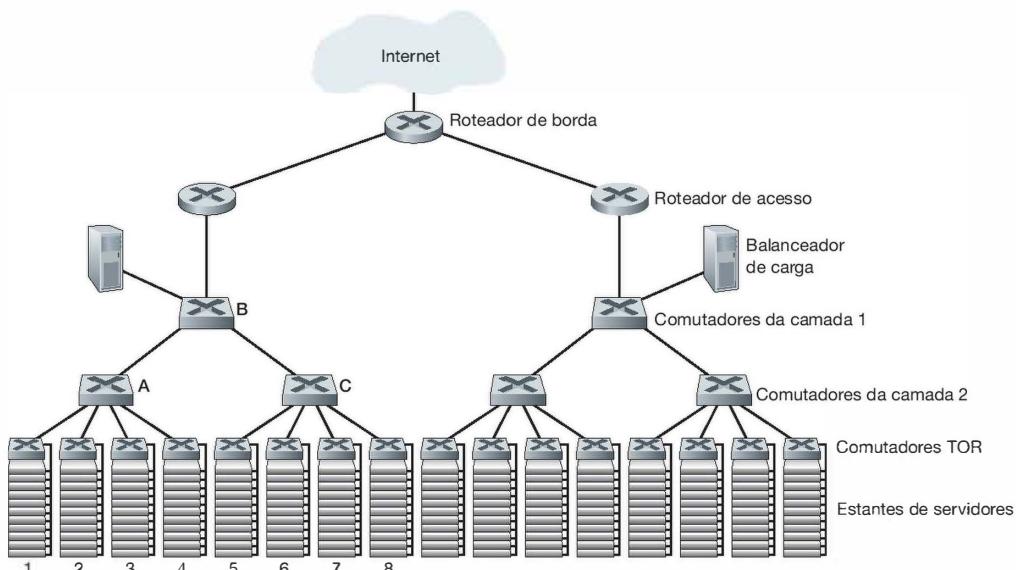
está na interessante posição de ter dois caminhos MPLS para chegar até A: por meio da interface 0 com rótulo MPLS de saída 10, e da interface 1 com um rótulo MPLS 8. O quadro geral apresentado na Figura 5.29 é que os dispositivos IP R5, R6, A e D estão conectados em conjunto via uma infraestrutura MPLS (roteadores habilitados a MPLS R1, R2, R3 e R4) praticamente do mesmo modo como uma LAN comutada ou uma rede ATM podem conectar dispositivos IP entre si. E, do mesmo modo que uma LAN comutada ou uma rede ATM, os roteadores R1 a R4 habilitados para MPLS *fazem isso sem jamais tocar o cabeçalho IP de um pacote.*

Nessa discussão, não especificamos o protocolo utilizado para distribuir rótulos entre roteadores habilitados para MPLS, pois os detalhes dessa sinalização estariam muito além do escopo deste livro. Observamos, entretanto, que o grupo de trabalho da IETF para o MPLS especificou no [RFC 3468] que uma extensão do protocolo RSVP, conhecida como RSVP-TE [RFC 3209], será o foco de seus esforços para a sinalização MPLS. Também não discutimos como o MPLS realmente calcula os caminhos para pacotes entre roteadores habilitados para MPLS, nem como ele reúne informações de estado do enlace (por exemplo, quantidade de largura de banda do enlace não reservada pelo MPLS) para usar nesses cálculos de caminho. Os algoritmos de roteamento de estado de enlace (por exemplo, OSPF) foram estendidos para inundar essa informação aos roteadores habilitados para MPLS. É interessante que os algoritmos reais de cálculo de caminho não são padronizados, e são atualmente específicos do fornecedor.

Até aqui, a ênfase de nossa discussão sobre o MPLS tem sido o fato de que esse protocolo executa comutação com base em rótulos, sem precisar considerar o endereço IP de um pacote. As verdadeiras vantagens do MPLS e a razão do atual interesse por ele, contudo, não estão nos aumentos substanciais nas velocidades de comutação, mas nas novas capacidades de gerenciamento de tráfego que o MPLS proporciona. Como já vimos, R4 tem *dois* caminhos MPLS até A. Se o repasse fosse executado até a camada IP tendo como base o endereço IP, os protocolos de roteamento IP que estudamos no Capítulo 4 especificariam um único caminho de menor custo até A. Assim, o MPLS provê a capacidade de repassar pacotes por rotas, o que não seria possível usando protocolos padronizados de roteamento IP. Essa é só uma forma simples de **engenharia de tráfego** usando o MPLS [RFC 3346; RFC 3272; RFC 2702; Xiao, 2000], com a qual um operador de rede pode suplantar o roteamento IP normal e obrigar que uma parte do tráfego dirigido a um dado destino siga por um caminho, e que outra parte do tráfego dirigido ao mesmo destino siga por outro (seja por política, por desempenho ou por alguma outra razão).

Também é possível utilizar MPLS para muitas outras finalidades. O protocolo pode ser usado para realizar restauração rápida de caminhos de repasse MPLS, por exemplo, mudar a rota do tráfego que passa por um caminho previamente calculado, restabelecido, em resposta à falha de enlace [Kar, 2000; Huang, 2002; RFC 3469]. Por fim, observamos que o MPLS pode ser, e tem sido, utilizado para implementar as denominadas **redes privadas virtuais** (*virtual private networks* — VPN). Ao executar uma VPN para um cliente, um ISP utiliza uma rede habilitada para MPLS para conectar as várias redes do cliente. O MPLS também pode ser usado para isolar os recursos e o endereçamento utilizados pela VPN do cliente dos outros usuários que estão cruzando a rede do IS; para obter mais detalhes, veja DeClercq [2002].

Nossa discussão sobre o MPLS foi necessariamente breve e aconselhamos o leitor a consultar as referências que mencionamos. Observamos que são tantas as utilizações possíveis do MPLS que ele parece estar se tornando o canivete suíço da engenharia de tráfego da Internet!

5.6 REDES DO DATACENTER

Nos últimos anos, empresas de Internet como Google, Microsoft, Facebook e Amazon (bem como suas equivalentes na Ásia e na Europa) construíram datacenters maciços, cada um abrigando dezenas a centenas de milhares de hospedeiros e dando suporte simultâneo a muitas aplicações distintas na nuvem (por exemplo, busca, correio eletrônico, redes sociais e comércio eletrônico). Cada datacenter tem sua própria **rede do datacenter** que interconecta seus hospedeiros e liga o datacenter à Internet. Nesta seção, faremos uma rápida introdução às redes do datacenter para aplicações de nuvem.

O custo de um grande datacenter é imenso, ultrapassando US$ 12 milhões por mês para um datacenter de 100 mil hospedeiros [Greenberg, 2009a]. Desses, 45% podem ser atribuídos aos próprios hospedeiros (que precisam ser substituídos a cada 3-4 anos); 25% à infraestrutura, incluindo transformadores, "*no-breaks*", geradores para faltas de energia prolongadas e sistemas de resfriamento; 15% para custos com consumo de energia elétrica; e 15% para redes, incluindo dispositivos (comutadores, roteadores e balanceadores de carga), enlaces externos e custos de tráfego de dados. (Nessas porcentagens, os custos com equipamento são amortizados, assim uma métrica de custo comum é aplicada para compras de única vez e despesas contínuas, como energia.) Embora o uso de redes não seja o maior dos custos, sua inovação é a chave para reduzir o custo geral e maximizar o desempenho [Greenberg, 2009a].

As abelhas trabalhadoras em um datacenter são os hospedeiros: eles servem o conteúdo (por exemplo, páginas Web e vídeos), armazenam mensagens de correio eletrônico e documentos, e realizam coletivamente cálculos maciçamente distribuídos (por exemplo, cálculos de índice distribuídos para mecanismos de busca). Os hospedeiros nos datacenters, chamados **lâminas** e semelhantes a embalagens de pizza, são em geral hospedeiros básicos incluindo CPU, memória e armazenamento de disco. Os hospedeiros são empilhados em estantes, com cada uma normalmente tendo de 20 a 40 lâminas. No topo de cada estante há um comutador, devidamente denominado **comutador do topo da estante (TOR —** *Top Of Rack*), que interconecta os hospedeiros entre si e com outros comutadores no datacenter. Especificamente, cada hospedeiro na estante tem uma placa de interface de rede que se conecta ao seu comutador TOR, e cada comutador TOR tem portas adicionais que podem ser conectadas a outros comutadores. Embora os hospedeiros de hoje geralmente tenham conexões Ethernet a 1 Gbit/s com seus comutadores TOR, conexões de 10 Gbits/s poderão se tornar comuns. Cada hospedeiro também recebe seu próprio endereço IP interno ao datacenter.

A rede do datacenter aceita dois tipos de tráfego: tráfego fluindo entre clientes externos e hospedeiros internos, e tráfego fluindo entre hospedeiros internos. Para tratar dos fluxos entre os clientes externos e os hospedeiros internos, a rede do datacenter inclui um ou mais **roteadores de borda**, conectando a rede do datacenter à Internet pública. Portanto, a rede do datacenter interconecta as estantes umas com as outras e conecta as estantes aos roteadores de borda. A Figura 5.30 mostra um exemplo de uma rede do datacenter. O **projeto de rede do**

FIGURA 5.30 UMA REDE DO DATACENTER COM UMA TOPOLOGIA HIERÁRQUICA

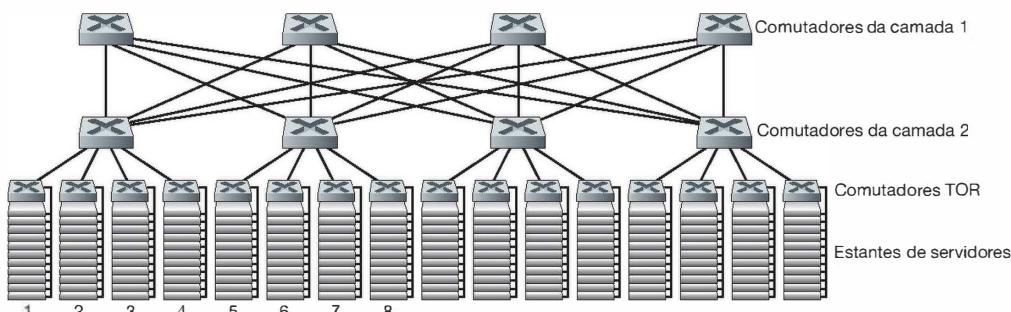
datacenter, a arte de projetar a rede de interconexão e os protocolos que conectam as estantes entre si e com os roteadores de borda, tornou-se um ramo importante da pesquisa sobre redes de computadores nos últimos anos [Al-Fares, 2008; Greenberg, 2009a; Greenberg, 2009b; Mydotr, 2009; Guo, 2009; Chen, 2010; Abu-Libdeh, 2010; Alizadeh, 2010; Wang, 2010; Farrington, 2010; Halperin, 2011; Wilson, 2011; Mudigonda, 2011; Ballani, 2011; Curtis, 2011; Raiciu, 2011].

Balanceamento de carga

Um datacenter de nuvem, como um datacenter da Google ou da Microsoft, oferece muitas aplicações simultaneamente, como aplicações de busca, correio eletrônico e vídeo. Para dar suporte a solicitações de clientes externos, cada aplicação é associada a um endereço IP publicamente visível, ao qual clientes enviam suas solicitações e do qual eles recebem respostas. Dentro do datacenter, as solicitações externas são direcionadas primeiro a um **balanceador de carga**, cuja função é distribuir as solicitações aos hospedeiros, equilibrando a carga entre os hospedeiros como uma função de sua carga atual. Um grande datacenter normalmente terá vários balanceadores de carga, cada um dedicado a um conjunto de aplicações de nuvem específicas. Esse balanceador de carga às vezes é conhecido como "comutador da camada 4", pois toma decisões com base no número da porta de destino (camada 4), bem como no endereço IP de destino no pacote. Ao receber uma solicitação por uma aplicação em particular, o balanceador de carga a encaminha para um dos hospedeiros que trata da aplicação. (Um hospedeiro pode, então, invocar os serviços de outros hospedeiros, para ajudar a processar a solicitação.) Quando o hospedeiro termina de processar a solicitação, ele envia sua resposta de volta ao balanceador de carga, que por sua vez repassa a resposta de volta ao cliente externo. O balanceador de carga não só equilibra a carga de trabalho entre os hospedeiros, mas também oferece uma função tipo NAT, traduzindo o endereço IP externo, público, para o endereço IP interno do hospedeiro apropriado, e depois traduzindo de volta os pacotes que trafegam na direção contrária, de volta aos clientes. Isso impede que os clientes entrem em contato direto com os hospedeiros, o que tem o benefício para a segurança de ocultar a estrutura de rede interna e impedir que clientes interajam diretamente com os hospedeiros.

Arquitetura hierárquica

Para um datacenter pequeno, abrigando apenas alguns milhares de hospedeiros, uma rede simples, que consiste em um roteador de borda, um balanceador de carga e algumas dezenas de estantes, todas interconectadas por um único comutador Ethernet, possivelmente seria suficiente. Mas, para escalar para dezenas a centenas de milhares de hospedeiros, um datacenter normalmente emprega uma **hierarquia de roteadores e comutadores**, como a topologia mostrada na Figura 5.30. No topo da hierarquia, o roteador de borda se conecta aos roteadores de acesso (somente dois aparecem na Figura 5.30, mas pode haver muito mais). Abaixo de cada roteador de acesso há três camadas de comutadores. Cada roteador de acesso se conecta a um comutador da camada superior, e cada comutador da camada superior se conecta a vários comutadores da segunda camada e a um balanceador de carga. Cada comutador da segunda camada, por sua vez, se conecta a várias estantes por meio dos comutadores TOR das estantes (comutadores da terceira camada). Todos os enlaces em geral utilizam Ethernet para seus protocolos da camada de enlace e da camada física, com uma mistura de cabeamento de cobre e fibra. Com esse projeto hierárquico, é possível escalar um datacenter até centenas de milhares de hospedeiros.

Como é crítico para um provedor de aplicação de nuvem oferecer aplicações continuamente com alta disponibilidade, os datacenters também incluem equipamento de rede redundante e enlaces redundantes em seus projetos (isso não está incluído na Figura 5.30). Por exemplo, cada comutador TOR pode se conectar a dois comutadores da camada 2, e cada roteador de acesso, comutador da camada 1 e comutador da camada 2 pode ser duplicado e integrado ao projeto [Cisco, 2012; Greenberg, 2009b]. No projeto hierárquico da Figura 5.30, observe que os hospedeiros abaixo de cada roteador de acesso formam uma única sub-rede. Para localizar o tráfego de difusão ARP, cada uma dessas sub-redes é dividida ainda mais em sub-redes de VLAN menores, cada uma compreendendo algumas centenas de hospedeiros [Greenberg, 2009a].

Embora a arquitetura hierárquica convencional que acabamos de descrever resolva o problema de escala, ela sofre de *capacidade limitada de hospedeiro-a-hospedeiro* [Greenberg, 2009b]. Para entender essa limitação, considere novamente a Figura 5.30 e suponha que cada hospedeiro se conecte ao seu comutador TOR com um enlace de 1 Gbit/s, enquanto os enlaces entre os comutadores são enlaces Ethernet de 10 Gbits/s. Dois hospedeiros na mesma estante sempre podem se comunicar com 1 Gbit/s completo, limitados apenas pela velocidade das placas de interface de rede dos hospedeiros. Porém, se houver muitos fluxos simultâneos na rede do datacenter, a velocidade máxima entre dois hospedeiros em estantes *diferentes* pode ser muito menor. Para ter uma ideia desse problema, considere um padrão de tráfego consistindo em 40 fluxos simultâneos entre 40 pares de hospedeiros em diferentes estantes. Especificamente, suponha que cada um dos 10 hospedeiros na estante 1 da Figura 5.30 envie um fluxo a um hospedeiro correspondente na estante 5. De modo semelhante, há dez fluxos simultâneos entre pares de hospedeiros nas estantes 2 e 6, dez fluxos simultâneos entre as estantes 3 e 7, e dez fluxos simultâneos entre as estantes 4 e 8. Se cada fluxo compartilha uniformemente a capacidade de um enlace com outros fluxos atravessando esse enlace, então os 40 fluxos cruzando o enlace de 10 Gbits/s de A-para-B (bem como o enlace de 10 Gbits/s de B-para-C) receberão, cada um, apenas 10 Gbits/s / 40 = 250 Mbits/s, que é muito menor do que a velocidade de 1 Gbit/s da placa de interface de rede. O problema se torna ainda mais grave para fluxos entre hospedeiros que precisam trafegar por uma camada mais alta na hierarquia. Uma solução possível para essa limitação é empregar comutadores e roteadores com velocidade mais alta. Mas isso aumentaria significativamente o custo do datacenter, pois os comutadores e roteadores com grandes velocidades de porta são muito caros.

O suporte à comunicação com alta largura de banda de hospedeiro-a-hospedeiro é importante porque um requisito básico nos datacenters é a flexibilidade no posicionamento de computação e serviços [Greenberg, 2009b; Farrington, 2010]. Por exemplo, um mecanismo de busca da Internet em grande escala pode ser executado em milhares de hospedeiros espalhados por várias estantes com requisitos de largura de banda significativos entre todos os pares de hospedeiros. De modo semelhante, um serviço de computação de nuvem, como EC2, pode querer colocar as diversas máquinas virtuais que compreendem um serviço do cliente nos hospedeiros físicos com mais capacidades, independentemente do seu local no datacenter. Se esses hospedeiros físicos estiverem espalhados por várias estantes, gargalos na rede, como descrevemos no parágrafo anterior, podem ocasionar um desempenho fraco.

Tendências para as redes no datacenter

Para reduzir o custo dos datacenters, e ao mesmo tempo melhorar seu atraso e vazão, gigantes de nuvem da Internet, como Google, Facebook, Amazon e Microsoft, estão continuamente implantando novos projetos de rede do datacenter. Embora esses projetos sejam próprios, mesmo assim, muitas tendências importantes podem ser identificadas.

FIGURA 5.31 TOPOLOGIA DE REDE DE DADOS ALTAMENTE INTERCONECTADA

Uma dessas tendências é executar novas arquiteturas de interconexão e protocolos de rede que contornem as desvantagens dos projetos hierárquicos tradicionais. Uma tática desse tipo é substituir a hierarquia de comutadores e roteadores por uma **topologia totalmente conectada** [Al-Fares, 2008; Greenberg, 2009b; Guo, 2009], como aquela mostrada na Figura 5.31. Nesse projeto, cada comutador da camada 1 se conecta a todos os comutadores da camada 2, de modo que (1) o tráfego de hospedeiro-a-hospedeiro nunca precisa subir além das camadas de comutadores e (2) com *n* comutadores da camada 1, entre dois quaisquer comutadores da camada 2, existem *n* caminhos separados. Esse projeto pode melhorar significativamente a capacidade de hospedeiro-a-hospedeiro. Para ver isso, considere novamente nosso exemplo de 40 fluxos. A topologia na Figura 5.31 pode lidar com tal padrão de fluxos, pois há quatro caminhos distintos entre o primeiro comutador da camada 2 e o segundo comutador dessa camada, oferecendo em conjunto uma capacidade agregada de 40 Gbits/s entre os dois primeiros comutadores da camada 2. Tal projeto não só alivia a limitação de capacidade de hospedeiro-a-hospedeiro, mas também cria um ambiente de computação e serviço mais flexível, no qual a comunicação entre duas estantes quaisquer não conectadas ao mesmo comutador é logicamente equivalente, independentemente de seus locais no datacenter.

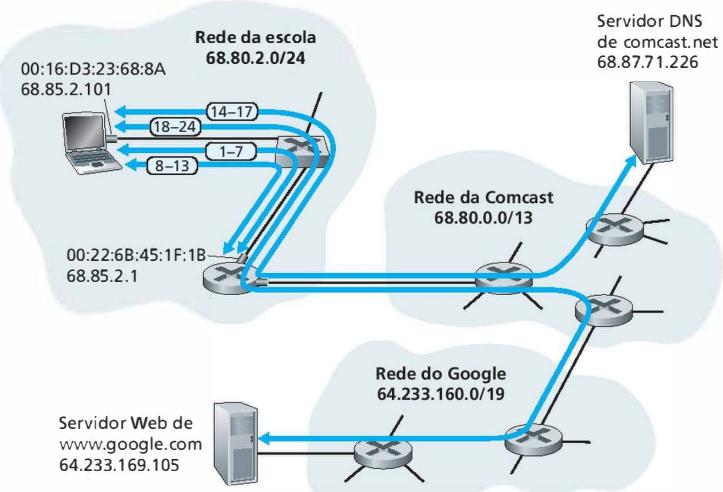
Outra tendência importante é empregar datacenters modulares (MDCs) baseados em contêineres [YouTube, 2009; Waldrop, 2007]. Em um MDC, uma fábrica monta, dentro de um contêiner de navio padrão de 12 m, um "mini-datacenter" e envia o contêiner ao local do datacenter. Cada contêiner tem até alguns milhares de hospedeiros, empilhados em dezenas de estantes, que são dispostas próximas umas das outras. No local do datacenter, vários contêineres são interconectados entre si e também com a Internet. Quando um contêiner pré-fabricado é implantado em um datacenter, normalmente, é difícil dar assistência técnica. Assim, cada contêiner é projetado para realizar a degradação de desempenho controlada: quando os componentes (servidores e comutadores) falham com o tempo, ele continua a operar, mas com um desempenho diminuído. Quando muitos componentes tiverem falhado e o desempenho tiver caído abaixo de um patamar, o contêiner inteiro é removido e substituído por um novo.

A montagem de um datacenter a partir de contêineres cria novos desafios de rede. Com um MDC, existem dois tipos de redes: as redes internas ao contêiner, dentro de cada contêiner, e a rede central conectando cada contêiner [Guo, 2009; Farrington, 2010]. Dentro de cada um, na escala de até alguns milhares de hospedeiros, é possível montar uma rede totalmente conectada (como já explicamos) usando comutadores Gigabit Ethernet comuns, pouco dispendiosos. Porém, o projeto da rede central, que interconecta centenas a milhares de contêineres enquanto oferece alta largura de banda de hospedeiro-a-hospedeiro entre os contêineres para cargas de trabalho típica, continua sendo um problema desafiador. Uma arquitetura de comutador híbrido eletro-ótico para interconectar os contêineres é proposta em Farrington [2010].

Ao usar topologias altamente interconectadas, um dos principais problemas é o projeto de algoritmos de roteamento entre os comutadores. Uma possibilidade [Greenberg, 2009b] é usar uma forma de roteamento aleatório. Outra possibilidade [Guo, 2009] é implementar múltiplas placas de interface de rede em cada hospedeiro, conectar cada um a vários comutadores de baixo custo e permitir que os próprios hospedeiros direcionem o tráfego de modo inteligente entre os comutadores. Hoje, variações e extensões dessas técnicas estão sendo executadas em datacenters contemporâneos. Muito mais inovações no projeto de datacenter provavelmente surgirão; os leitores interessados poderão ler os muitos artigos recentes sobre projeto de redes do datacenter.

5.7 UM DIA NA VIDA DE UMA SOLICITAÇÃO DE PÁGINA WEB

Agora que já cobrimos a camada de enlace neste capítulo, e da rede, de transporte e a camada de aplicação em capítulos anteriores, nossa jornada pela pilha de protocolos está completa! Bem no início deste livro (Seção 1.1), escrevemos que "grande parte deste livro trata de protocolos de redes de computadores" e, nos primeiros cinco capítulos, vimos que de fato isso é verdade. Antes de nos dirigirmos aos tópicos dos próximos capítulos, gostaríamos de finalizar nossa jornada pela pilha de protocolos considerando uma visão integrada e holística

dos que vimos até agora. Uma forma de termos essa visão geral é identificarmos os vários (vários!) protocolos envolvidos na satisfação de simples pedidos, como fazer o download de uma página Web. A Figura 5.32 ilustra a imagem que queremos passar: um estudante, Bob, conecta seu notebook ao comutador Ethernet da sua escola e faz o download de uma página Web (digamos que é a página principal de www.google.com). Como já sabemos, existe *muito* mais sob a superfície do que se imagina para realizar esta solicitação que parece simples. O laboratório Wireshark, ao final deste capítulo, examina cenários de comunicação mais em detalhes, contendo vários pacotes envolvidos em situações parecidas.

5.7.1 Começando: DHCP, UDP, IP e Ethernet

Suponha que Bob liga seu notebook e o conecta via um cabo Ethernet ao comutador Ethernet da escola, que por sua vez é conectado ao roteador da escola, como demonstrado na Figura 5.32. O roteador da escola é conectado a um ISP, que neste exemplo é comcast.net. Neste exemplo, a comcast.net fornece o serviço DNS para a escola; dessa forma, o servidor DNS se localiza em uma rede da Comcast, e não na rede da escola. Vamos supor que servidor DHCP está sendo executado dentro do roteador, como costuma acontecer.

Quando Bob conecta seu notebook à rede pela primeira vez, não consegue fazer nada (por exemplo, baixar uma página Web) sem um endereço IP. Assim, a primeira ação relacionada à rede, tomada pelo notebook, é executar o protocolo DHCP para obter um endereço IP, bem como outras informações do servidor DHCP local:

1. O sistema operacional do notebook de Bob cria uma mensagem de solicitação DHCP (Seção 4.4.2) e a coloca dentro do segmento UDP (Seção 3.3) com a porta de destino 67 (servidor DHCP) e porta de origem 68 (cliente DHCP). O segmento é então colocado dentro de um datagrama IP (Seção 4.4.1) com um endereço IP de destino por difusão (255.255.255.255) e um endereço IP de origem 0.0.0.0, já que o notebook do Bob ainda não tem um endereço IP.
2. O datagrama IP contendo uma mensagem de solicitação DHCP é colocado dentro de um quadro Ethernet (Seção 5.4.2). O quadro Ethernet tem endereços de destino MAC FF:FF:FF:FF:FF:FF de modo que o quadro será transmitido a todos os dispositivos conectados ao comutador (onde se espera que haja um servidor DHCP); o quadro de origem do endereço MAC é do notebook do Bob, 00:16:D3:23:68:8A.

FIGURA 5.32 UM DIA NA VIDA DE UMA SOLICITAÇÃO DE PÁGINA WEB: PREPARAÇÃO E AÇÕES DA REDE

3. O quadro de difusão Ethernet contendo a solicitação DHCP é o primeiro a ser enviado pelo notebook de Bob para o comutador Ethernet. O comutador transmite o quadro da entrada para todas as portas de saída, incluindo a porta conectada ao roteador.
4. O roteador recebe o quadro Ethernet transmitido, que contém a solicitação DHCP na sua interface com endereço MAC 00:22:6B:45:1F:1B, e o datagrama IP é extraído do quadro Ethernet. O endereço IP de destino transmitido indica que este datagrama IP deveria ser processado por protocolos de camadas mais elevadas em seu nó, de modo que, dessa forma, a carga útil do datagrama (um segmento UDP) é demultiplexada (Seção 3.2) até o UDP, e a mensagem de solicitação é extraída do segmento UDP. Agora o servidor DHCP tem a mensagem de solicitação DHCP.
5. Suponhamos que o servidor DHCP que esteja sendo executado dentro de um roteador possa alocar o endereço IP no bloco CIDR (Seção 4.4.2) 68.85.2.0/24. Neste exemplo, todos os endereços IP usados na escola estão dentro do bloco de endereços da Comcast. Vamos supor que o servidor DHCP designe o endereço 68.85.2.101 ao notebook do Bob. O servidor DHCP cria uma mensagem de ACK DHCP (Seção 4.4.2) contendo um endereço IP, assim como o endereço IP do servidor DNS (68.87.71.226), o endereço IP para o roteador de borda (*gateway*) *default* (68.85.2.1) e o bloco de sub-rede (68.85.2.0/24) (equivalente à "máscara de rede"). A mensagem DHCP é colocada dentro de um segmento UDP, o qual é colocado dentro de um datagrama IP, o qual é colocado dentro de um quadro Ethernet. O quadro Ethernet tem o endereço MAC de origem da interface do roteador na rede doméstica (00:22:6B:45:1F:1B) e um endereço MAC de destino do notebook do Bob (00:16:D3:23:68:8A).
6. O quadro Ethernet contendo o ACK DHCP é enviado (individualmente) pelo roteador ao comutador. Uma vez que o comutador realiza a autoaprendizagem (Seção 5.4.3) e que recebe previamente um quadro do notebook de Bob (que contém a solicitação DHCP), o comutador sabe como encaminhar um quadro endereçado a 00:16:D3:23:68:8A apenas para a porta de saída que leva ao notebook do Bob.
7. O notebook do Bob recebe o quadro Ethernet que contém o ACK DHCP, extrai o datagrama IP do quadro Ethernet, extrai o segmento UDP do datagrama IP, e extrai a mensagem ACK DHCP do segmento UDP. Então, o cliente DHCP do Bob grava seu endereço IP e o endereço IP do seu servidor DNS. Ele também instala o endereço do roteador de borda *default* em sua tabela de repasse de IP (Seção 4.1). O notebook de Bob enviará todos os datagramas com endereços de destino fora de sua sub-rede 68.85.2.0/24 à saída-padrão. Nesse momento, o notebook do Bob inicializou os seus componentes de rede e está pronto para começar a processar a busca da página Web. (Observe que apenas as duas últimas etapas DHCP das quatro apresentadas no Capítulo 4 são de fato necessárias.)

5.7.2 Ainda começando: DNS, ARP

Quando Bob digita o URL www.google.com em seu navegador, ele inicia uma longa cadeia de eventos que no fim resultarão na exibição da página principal do Google no navegador. O navegador de Bob inicia o processo ao criar um **socket TCP** (Seção 2.7) que será usado para enviar uma **requisição HTTP** (Seção 2.2) para www.google.com. Para criar o *socket*, o notebook de Bob precisará saber o endereço IP de www.google.com. Aprendemos, na Seção 2.5, que o **protocolo DNS** é usado para fornecer serviços de tradução de nomes para endereço IP.

8. O sistema operacional do notebook de Bob cria então uma mensagem de consulta DNS (Seção 2.5.3), colocando a cadeia de caracteres "www.google.com" no campo de pergunta da mensagem DNS. Essa mensagem é então colocada dentro de um segmento UDP, com a porta de destino 53 (servidor DNS). O segmento UDP é então colocado dentro de um datagrama IP com um endereço de destino IP 68.87.71.226 (o endereço do servidor DNS retomado pelo ACK DHCP na etapa 5) e um endereço IP de origem 68.85.2.101.
9. O notebook de Bob coloca então um datagrama contendo a mensagem de consulta DNS em um quadro Ethernet. Este quadro será enviado (endereçado, na camada de enlace) ao roteador de borda da rede da escola de Bob. No entanto, apesar de o notebook de Bob conhecer o endereço IP do roteador de borda da rede da escola (68.85.2.1), via mensagem ACK DHCP na etapa 5 anterior, ele não sabe o endereço MAC

do roteador de borda. Para que o notebook do Bob obtenha o endereço MAC do roteador de borda, ele precisará usar o protocolo ARP (Seção 5.4.1).

10. O notebook de Bob cria uma mensagem de consulta ARP direcionada ao endereço IP 68.85.2.1 (a porta-padrão), coloca a mensagem ARP dentro do quadro Ethernet para ser transmitido por difusão ao endereço de destino (FF:FF:FF:FF:FF:FF) e envia o quadro Ethernet ao comutador, que entrega o quadro a todos os dispositivos, incluindo o roteador de borda.
11. O roteador de borda recebe um quadro contendo a mensagem de consulta ARP na interface da rede da escola, e descobre que o endereço IP de destino 68.85.2.1 na mensagem ARP corresponde ao endereço IP de sua interface. Então, o roteador de borda prepara uma **resposta ARP**, indicando que o seu endereço MAC 00:22:6B:45:1F:1B corresponde ao endereço IP 68.85.2.1. Ele coloca a mensagem de resposta ARP em um quadro Ethernet, com o endereço de destino 00:16:D3:23:68:8A (notebook do Bob) e envia o quadro ao comutador, que entrega o quadro ao notebook de Bob.
12. O notebook de Bob recebe o quadro que contém a mensagem de resposta ARP e extrai o endereço MAC do roteador de borda (00:22:6B:45:1F:1B) da mensagem de resposta ARP.
13. O notebook de Bob pode agora (*enfim!*) endereçar o quadro Ethernet com a mensagem de consulta DNS ao endereço MAC do roteador de borda. Observe que o datagrama nesse quadro tem o endereço IP de destino 68.87.71.226 (servidor DNS), enquanto o quadro tem o endereço de destino 00:22:6B:45:1F:1B (roteador de borda). O notebook de Bob envia esse quadro ao comutador, que entrega o quadro ao roteador de borda.

5.7.3 Ainda começando: roteamento intradomínio ao servidor DNS

14. O roteador de borda recebe o quadro e extrai o datagrama IP que contém a consulta DNS. O roteador procura o endereço de destino desse datagrama (68.87.71.226) e determina de sua tabela de repasse que ele deve ser enviado ao roteador da extremidade esquerda na rede Comcast, como na Figura 5.32. O datagrama IP é colocado em um quadro de uma camada de enlace apropriado ao enlace conectando o roteador da escola ao roteador Comcast da extremidade esquerda, e o quadro é enviado através desse enlace.
15. O roteador da extremidade esquerda na rede Comcast recebe o quadro, extrai o datagrama IP, examina o endereço de destino do datagrama (68.87.71.226) e determina a interface de saída pela qual enviará o datagrama ao servidor DNS de sua tabela de repasse, que foi preenchida pelo protocolo intradomínio da Comcast (como **RIP**, **OSPF** ou **IS-IS**, Seção 4.6), assim como o **protocolo intradomínio da Internet**, **BGP**.
16. Por fim, o datagrama IP contendo a consulta DNS chega ao servidor DNS. Este extrai a mensagem de consulta DNS, procura o nome em www.google.com na sua base de dados DNS (Seção 2.5), e encontra o **registro de recurso DNS** que contém o endereço IP (64.233.169.105) para www.google.com (supondo-se que está em *cache* no servidor DNS). Lembre-se que este dado em *cache* foi originado no **servidor DNS com autoridade** (Seção 2.5.2) para google.com. O servidor DNS forma uma **mensagem DNS de resposta** contendo o mapeamento entre nome de hospedeiro e endereço IP, e coloca a mensagem DNS de resposta em um segmento UDP, e o segmento dentro do datagrama IP endereçado ao notebook do Bob (68.85.2.101). Esse datagrama será encaminhado de volta ao roteador da escolha por meio da rede Comcast, e de lá ao notebook de Bob, via comutador Ethernet.
17. O notebook de Bob extrai o endereço IP do servidor www.google.com da mensagem DNS. *Enfim*, depois de *muito* trabalho, o notebook de Bob está pronto para contatar o servidor www.google.com!

5.7.4 Interação cliente-servidor Web: TCP e HTTP

18. Agora que o notebook de Bob tem o endereço IP de www.google.com, ele está pronto para criar um ***socket* TCP** (Seção 2.7), que será usado para enviar uma mensagem **HTTP GET** (Seção 2.2.3) para www.google.com.

Quando Bob cria um *socket* TCP, o TCP de seu notebook precisa primeiro executar uma **apresentação de três vias** (Seção 3.5.6) com o TCP em www.google.com. Então, o notebook de Bob primeiro cria um segmento **TCP SYN** com a porta de destino 80 (para HTTP), coloca o segmento TCP dentro de um datagrama IP, com o endereço IP de destino 64.233.169.105 (www.google.com), coloca o datagrama dentro de um quadro com o endereço de destino 00:22:6B:45:1F:1B (o roteador de borda) e envia o quadro ao comutador.

19. Os roteadores da rede da escola, da rede Comcast e da rede do Google encaminham o datagrama contendo o TCP SYN até www.google.com, usando a tabela de repasse em cada roteador, como nas etapas 14-16. Os itens da tabela de repasse controlando o envio de pacotes interdomínio entre as redes da Comcast e do Google são determinados pelo protocolo **BGP** (Seção 4.6.3).
20. Por fim, o datagrama contendo o TCP SYN chega em www.google.com. A mensagem TCP SYN é extraída do datagrama e demultiplexada ao *socket* associado à porta 80. Um *socket* de conexão (Seção 2.7) é criado para a conexão TCP entre o servidor HTTP do Google e o notebook de Bob. Um segmento TCP SYNACK (Seção 3.5.6) é gerado, colocado dentro de um datagrama endereçado ao notebook de Bob, e enfim colocado em um quadro da camada de enlace apropriado ao enlace conectando www.google.com ao roteador de primeiro salto.
21. O datagrama que contém o segmento TCP SYNACK é encaminhado através das redes do Google, Comcast e da escola, finalmente chegando ao cartão Ethernet no computador de Bob. O datagrama é demultiplexado dentro do sistema operacional e entregue ao *socket* TCP criado na etapa 18, que entra em estado de conexão.
22. Agora, com o *socket* dentro do notebook de Bob (*finalmente!*), pronto para enviar bytes a www.google.com, o navegador cria uma mensagem HTTP GET (Seção 2.2.3) contendo a URL a ser procurada. A mensagem HTTP GET é enviada ao *socket*, com a mensagem GET se tornando a carga útil do segmento TCP. O segmento TCP é colocado em um datagrama e enviado e entregue em www.google.com, como nas etapas 18-20.
23. O servidor HTTP www.google.com lê a mensagem HTTP GET do *socket* TCP, cria uma mensagem de **resposta HTTP** (Seção 2.2), coloca o conteúdo da página Web requisitada no corpo da mensagem de resposta HTTP, e envia a mensagem pelo *socket* TCP.
24. O datagrama contendo a mensagem de resposta HTTP é encaminhado através das redes do Google, da Comcast e da escola e chega ao notebook de Bob. O programa do navegador de Bob lê a resposta HTTP do *socket*, extrai o html da página do corpo da resposta HTTP, e enfim (*enfim!*) mostra a página Web!

Nosso cenário abrangeu muito do fundamento das redes de comunicação! Se você entendeu a maior parte da representação, então também viu muito do fundamento desde que leu a Seção 1.1, onde escrevemos "grande parte deste livro trata de protocolos de redes de computadores" e você pode ter se perguntado o que na verdade era um protocolo! Por mais detalhado que o exemplo possa parecer, nós omitimos uma série de protocolos possíveis (por exemplo, NAT executado no roteador de borda da escola, acesso sem fio à rede da escola, protocolos de segurança para acessar a rede da escola, ou segmentos ou datagramas codificados), e considerações (*cache* da Web, hierarquia DNS) que poderíamos encontrar na Internet pública. Estudaremos a maioria desses tópicos na segunda parte deste livro.

Por fim, observamos que nosso exemplo era integrado e holístico, mas também muito resumido de muitos protocolos que estudamos na primeira parte do livro. Este exemplo é mais focado nos aspectos de "como" e não no "por quê". Para obter uma visão mais ampla e reflexiva dos protocolos de rede em geral, veja Clark [1988]; [RFC 5218].

5.8 RESUMO

Neste capítulo, examinamos a camada de enlace — seus serviços, os princípios subjacentes à sua operação e vários protocolos específicos importantes que usam tais princípios na execução dos serviços da camada de enlace.

Vimos que o serviço básico é mover um datagrama da camada de rede de um nó (hospedeiro, comutador, roteador, ponto de acesso Wi-Fi) até um nó adjacente. Vimos também que todos os protocolos da camada de enlace operam encapsulando um datagrama da camada de rede dentro de um quadro da camada de enlace antes de transmitir o quadro pelo enlace até o nó adjacente. Além dessa função comum de enquadramento, contudo, aprendemos que diferentes protocolos da camada de enlace oferecem serviços muito diferentes de acesso ao enlace, entrega e transmissão. Essas diferenças decorrem, em parte, da vasta variedade de tipos de enlaces sobre os quais os protocolos de enlace devem operar. Um enlace ponto a ponto simples tem um único remetente e um único receptor comunicando-se por um único "fio". Um enlace de acesso múltiplo é compartilhado por muitos remetentes e receptores; por conseguinte, o protocolo da camada de enlace para um canal de acesso múltiplo tem um protocolo (seu protocolo de acesso múltiplo) para coordenar o acesso ao enlace. No caso de MPLS, o "enlace" que conecta dois nós adjacentes (por exemplo, dois roteadores IP adjacentes no sentido do IP — ou seja, são roteadores IP do próximo salto na direção do destino) pode, na realidade, constituir uma *rede* em si e por si próprio. Em certo sentido, a ideia de uma rede ser considerada um "enlace" não deveria parecer estranha. Um enlace telefônico que conecta um modem/computador residencial a um modem/roteador remoto, por exemplo, na verdade é um caminho que atravessa uma sofisticada e complexa *rede* telefônica.

Dentre os princípios subjacentes à comunicação por camada de enlace, examinamos técnicas de detecção e correção de erros, protocolos de acesso múltiplo, endereçamento da camada de enlace, virtualização (VLANs) e a construção de redes locais comutadas estendidas e redes do datacenter. Grande parte do foco atual na camada de enlace está sobre essas redes comutadas. No caso da detecção/correção de erros, examinamos como é possível adicionar bits ao cabeçalho de um quadro para detectar e algumas vezes corrigir erros de mudança de bits que podem ocorrer quando o quadro é transmitido pelo enlace. Analisamos esquemas simples de paridade e de soma de verificação, bem como o esquema mais robusto de verificação da redundância cíclica. Passamos, então, para o tópico dos protocolos de acesso múltiplo. Identificamos e estudamos três métodos amplos para coordenar o acesso a um canal de difusão: métodos de divisão de canal (TDM, FDM), métodos de acesso aleatório (os protocolos ALOHA e os CSMA) e métodos de revezamento (*polling* e passagem de permissão). Estudamos a rede de acesso a cabo e descobrimos que ela usa muitos desses métodos de acesso múltiplo. Vimos que uma consequência do compartilhamento de um canal de difusão por vários nós é a necessidade de prover endereços aos nós no nível da camada de enlace. Aprendemos que endereços da camada de enlace são bastante diferentes dos da camada de rede e que, no caso da Internet, um protocolo especial (o ARP — protocolo de resolução de endereço) é usado para fazer o mapeamento entre esses dois modos de endereçamento e estudamos o protocolo Ethernet, tremendamente bem-sucedido, com detalhes. Em seguida, examinamos como os nós que compartilham um canal de difusão formam uma LAN e como várias LANs podem ser conectadas para formar uma LAN de maior porte — tudo isso *sem* a intervenção do roteamento da camada de rede para a interconexão desses nós locais. Também aprendemos como múltiplas LANs virtuais podem ser criadas sobre uma única infraestrutura física de LAN.

Encerramos nosso estudo da camada de enlace focalizando como redes MPLS fornecem serviços da camada de enlace quando interconectadas com roteadores IP e com uma visão geral dos projetos de rede para os maciços datacenters atuais. Concluímos este capítulo (e, sem dúvida, os cinco primeiros) identificando os muitos protocolos que são necessários para buscar uma simples página Web. Com isso, *concluímos nossa jornada pela pilha de protocolos*! É claro que a camada física fica abaixo da de enlace, mas provavelmente será melhor deixar os detalhes da camada física para outro curso. Contudo, discutimos brevemente vários aspectos da camada física neste capítulo e no Capítulo 1 (nossa discussão sobre meios físicos na Seção 1.2). Consideraremos novamente a camada física quando estudarmos as características dos enlaces sem fio no próximo capítulo.

Embora nossa jornada pela pilha de protocolos esteja encerrada, o estudo sobre rede de computadores ainda não chegou ao fim. Nos quatro capítulos seguintes, examinaremos redes sem fio, redes multimídia, segurança da rede e gerenciamento de redes. Esses quatro tópicos não se encaixam perfeitamente em uma única camada; na verdade, cada um atravessa muitas camadas. Assim, entender esses tópicos (às vezes tachados de "tópicos avançados" em alguns textos sobre redes) requer uma boa base sobre todas as camadas da pilha de protocolos — uma base que se completou com nosso estudo sobre a camada de enlace de dados!

EXERCÍCIOS DE FIXAÇÃO E PERGUNTAS

Questões de revisão do Capítulo 5

SEÇÕES 5.1–5.2

R1. Considere a analogia de transporte na Seção 5.1.1. Se o passageiro é comparado com o datagrama, o que é comparado com o quadro da camada de enlace?

R2. Se todos os enlaces da Internet fornecessem serviço de entrega confiável, o serviço de entrega confiável do TCP seria redundante? Justifique sua resposta.

R3. Quais alguns possíveis serviços um protocolo da camada de enlace pode oferecer à camada de rede? Quais dos serviços da camada de enlace têm correspondentes no IP? E no TCP?

SEÇÃO 5.3

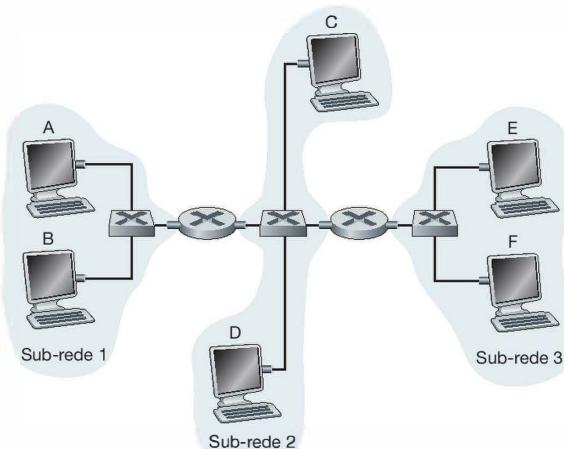
R4. Suponha que dois nós comecem a transmitir ao mesmo tempo um pacote de comprimento L por um canal *broadcast* de velocidade R. Denote o atraso de propagação entre os dois nós como d_{prop}. Haverá uma colisão se $d_{prop} < L/R$? Por quê?

R5. Na Seção 5.3, relacionamos quatro características desejáveis de um canal de difusão. O *slotted* ALOHA tem quais dessas características? E o protocolo de passagem de permissão, tem quais dessas características?

R6. No CSMA/CD, depois da quinta colisão, qual é a probabilidade de um nó escolher $K = 4$? O resultado $K = 4$ corresponde a um atraso de quantos segundos em uma Ethernet de 10 Mbits/s?

R7. Descreva os protocolos de *polling* e de passagem de permissão usando a analogia com as interações ocorridas em um coquetel.

R8. Por que o protocolo de passagem de permissão seria ineficiente se uma LAN tivesse um perímetro muito grande?

SEÇÃO 5.4

R9. Que tamanho tem o espaço de endereços MAC? E o espaço de endereços IPv4? E o espaço de endereços IPv6?

R10. Suponha que cada um dos nós A, B e C esteja ligado à mesma LAN de difusão (por meio de seus adaptadores). Se A enviar milhares de datagramas IP a B com quadro de encapsulamento endereçado ao endereço MAC de B, o adaptador de C processará esses quadros? Se processar, ele passará os datagramas IP desses quadros para C? O que mudaria em suas respostas se A enviasse quadros com o endereço MAC de difusão?

R11. Por que uma pesquisa ARP é enviada dentro de um quadro de difusão? Por que uma resposta ARP é enviada em um quadro com um endereço MAC de destino específico?

R12. Na rede da Figura 5.19, o roteador tem dois módulos ARP, cada um com sua própria tabela ARP. É possível que o mesmo endereço MAC apareça em ambas?

R13. Compare as estruturas de quadro das redes 10BASE-T, 100BASE-T e Gigabit Ethernet. Quais as diferenças entre elas?

R14. Considere a Figura 5.15. Quantas sub-redes existem no sentido de endereçamento da Seção 4.4?

R15. Qual o número máximo de VLANs que podem ser configuradas em um comutador que suporta o protocolo 802.1Q? Por quê?

R16. Imagine que N comutadores que suportam K grupos de VLAN serão conectados por meio de um protocolo de entroncamento. Quantas portas serão necessárias para conectar os comutadores? Justifique sua resposta.

PROBLEMAS

P1. Suponha que o conteúdo de informação de um pacote seja o padrão de bits 1110 0110 1001 1101 e que um esquema de paridade par esteja sendo usado. Qual seria o valor do campo de soma de verificação para o caso de um esquema de paridade bidimensional? Sua resposta deve ser tal que seja usado um campo de soma de verificação de comprimento mínimo.

P2. Dê um exemplo (que não seja o da Figura 5.5) mostrando que verificações de paridade bidimensional podem corrigir e detectar um erro de bit único. Dê outro exemplo mostrando um erro de bit duplo que pode ser detectado, mas não corrigido.

P3. Suponha que a parte da informação de um pacote (*D* da Figura 5.3) contenha 10 bytes consistindo na representação ASCII binária (8 bits) sem sinal da cadeia de caracteres "Networking". Calcule a soma de verificação da Internet para esses dados.

P4. Considere o problema anterior, mas suponha desta vez que esses 10 bytes contenham:

a. A representação binária dos números de 1 a 10.

b. A representação ASCII das letras B até K (letras maiúsculas).

c. A representação ASCII das letras B até K (letras minúsculas).

Calcule a soma de verificação da Internet para esses dados.

P5. Considere o gerador de 7 bits G =10011 e suponha que *D* tenha o valor de 1010101010. Qual é o valor de *R*?

P6. Considere o problema acima, mas suponha que *D* tenha o valor de:

a. 1001010101.

b. 0101101010.

c. 1010100000.

P7. Neste problema, exploramos algumas propriedades de CRC. Para o gerador G (=1001) dado na Seção 5.2.3, responda às seguintes questões:

a. Por que ele pode detectar qualquer erro de bit único no dado D?

b. Pode esse G detectar qualquer número ímpar de erros de bit? Por quê?

P8. Na Seção 5.3, fornecemos um esboço da derivação da eficiência do *slotted* ALOHA. Neste problema, concluiremos a derivação.

a. Lembre-se de que, quando há *N* nós ativos, a eficiência do *slotted* ALOHA é $Np(1 - p)^{N-1}$. Ache o valor de p que maximize essa expressão.

b. Usando o valor de p encontrado em (a), ache a eficiência do *slotted* ALOHA fazendo que *N* tenda ao infinito. *Dica*: $(1 - 1/N)^N$ tende a $1/e$ quando *N* tende ao infinito.

P9. Mostre que a eficiência máxima do ALOHA puro é $1/(2e)$. *Obs.*: Este problema será fácil se você tiver concluído o anterior!

P10. Considere dois nós, A e B, que usem um protocolo *slotted* ALOHA para competir pelo canal. Suponha que o nó A tenha mais dados para transmitir do que o B, e a probabilidade de retransmissão do nó A, p_A, seja maior do que a de retransmissão do nó B, p_B.

a. Determine a fórmula para a vazão média do nó A. Qual é a eficiência total do protocolo com esses dois nós?

b. Se $p_A = 2p_B$, a vazão média do nó A é duas vezes maior do que a do nó B? Por quê? Se não, como escolher p_A e p_B para que isso aconteça?

c. No geral, suponha que haja *N* nós, e entre eles o nó A tem a probabilidade de retransmissão $2p$ e todos os outros têm a probabilidade de retransmissão p. Determine as expressões para computar a vazão média do nó A e de qualquer outro nó.

P11. Suponha que quatro nós ativos — nós A, B, C e D — estejam competindo pelo acesso a um canal usando o *slotted* ALOHA. Imagine que cada nó tenha um número infinito de pacotes para enviar. Cada nó tenta transmitir em cada intervalo (*slot*) com probabilidade p. O primeiro é numerado como 1, o segundo como 2, e assim por diante.

a. Qual a probabilidade que o nó A tenha sucesso pela primeira vez no intervalo 5?

b. Qual a probabilidade que algum nó (A, B, C ou D) tenha sucesso no intervalo 4?

c. Qual a probabilidade que o primeiro sucesso ocorra no intervalo 3?

d. Qual a eficiência nesse sistema de quatro nós?

P12. Desenhe um gráfico da eficiência do *slotted* ALOHA e do ALOHA puro como uma função de p, para os seguintes valores de N:

a. $N = 15$.

b. $N = 25$.

c. $N = 35$.

P13. Considere um canal de difusão com N nós e uma taxa de transmissão de R bits/s. Suponha que o canal de difusão use o *polling* (com um nó de *polling* adicional) para acesso múltiplo. Imagine que o intervalo de tempo entre o momento em que o nó conclui a transmissão e o momento em que o nó subsequente é autorizado a transmitir (isto é, o atraso de *polling*) seja d_{poll}. Suponha ainda que, em uma rodada de *polling*, determinado nó seja autorizado a transmitir, no máximo, Q bits. Qual é a vazão máxima do canal de difusão?

P14. Considere três LANs interconectadas por dois roteadores, como mostrado na Figura 5.33.

a. Atribua endereços IP a todas as interfaces. Para a Sub-rede 1, use endereços do tipo 192.168.1.xxx; para a Sub-rede 2, use endereços do tipo 192.168.2.xxx, e para a Sub-rede 3 use endereços do tipo 192.168.3.xxx.

b. Atribua endereços MAC a todos os adaptadores.

c. Considere o envio de um datagrama IP do hospedeiro A ao hospedeiro F. Suponha que todas as tabelas ARP estejam atualizadas. Enumere todas as etapas, como foi feito no exemplo de um único roteador na Seção 5.4.1.

d. Repita (c), admitindo agora que a tabela ARP do hospedeiro remetente esteja vazia (e que as outras tabelas estejam atualizadas).

FIGURA 5.33 TRÊS SUB-REDES INTERCONECTADAS POR ROTEADORES

P15. Considere a Figura 5.33. Agora substituímos o roteador entre as sub-redes 1 e 2 pelo comutador S1, e chamamos de R1 o roteador entre as sub-redes 2 e 3.

a. Considere o envio de um datagrama IP do hospedeiro E ao hospedeiro F. O hospedeiro E pedirá ajuda ao roteador R1 para enviar o datagrama? Por quê? No quadro Ethernet que contém o datagrama IP, quais são os endereços IP e MAC de origem e de destino?

b. Suponha que E quisesse enviar um datagrama IP a B, e que o *cache* ARP de E não tenha o endereço MAC de B. E preparará uma consulta ARP para descobrir o endereço MAC de B? Por quê? No quadro Ethernet (que contém o datagrama IP destinado a B) entregue ao roteador R1, quais são os endereços de origem e destino IP e MAC?

c. Suponha que o hospedeiro A gostaria de enviar um datagrama IP ao hospedeiro B, e nem o *cache* ARP de A contém o endereço MAC de B, nem o *cache* ARP de B contém o endereço MAC de A. Suponha ainda que a tabela de encaminhamento do comutador S1 contenha entradas apenas para o hospedeiro B e para o roteador R1. Dessa forma, A transmitirá por difusão uma mensagem de requisição ARP. Que ações o comutador S1 tomará quando receber a mensagem de requisição ARP? O roteador R1 também receberá essa mensagem? Se sim, R1 a encaminhará para a Sub-rede 3? Assim que o hospedeiro B receber essa mensagem de requisição ARP, ele enviará a mensagem de resposta ARP de volta ao hospedeiro A. Mas enviará uma mensagem de consulta ARP para o endereço MAC de A? Por quê? O que o comutador S1 fará quando receber mensagem de resposta ARP do hospedeiro B?

P16. Considere o problema anterior, mas suponha que o roteador entre as sub-redes 2 e 3 é substituído por um comutador. Responda às questões de (a) a (c) do exercício anterior nesse novo contexto.

P17. Lembre-se de que, com o protocolo CSMA/CD, o adaptador espera $K \cdot 512$ tempos de bits após uma colisão, onde K é escolhido aleatoriamente. Para $K = 100$, quanto tempo o adaptador espera até voltar à etapa 2 para uma Ethernet de 10 Mbits/s? E para canal de difusão de 100 Mbits/s?

P18. Suponha que os nós A e B estejam no mesmo canal de difusão de 10 Mbits/s e que o atraso de propagação entre os dois nós seja de 325 tempos de bit. Suponha que pacotes CSMA/CD e Ethernet sejam usados para esse canal de difusão. Imagine que o nó A comece a transmitir um quadro e que, antes de terminar, o nó B comece a transmitir um quadro. O nó A pode terminar de transmitir antes de detectar que B transmitiu? Por quê? Se a resposta for sim, então A acredita, incorretamente, que seu quadro foi transmitido com sucesso, sem nenhuma colisão. *Dica*: suponha que no tempo $t = 0$ tempos de bit, A comece a transmitir um quadro. No pior dos casos, A transmite um quadro de tamanho mínimo de 512 + 64 tempos de bit. Portanto, A terminaria de transmitir o quadro em $t = 512 + 64$ tempos de bit. Então, a resposta será não, se o sinal de B chegar a A antes do tempo de bit $t = 512 + 64$ bits. No pior dos casos, quando o sinal de B chega a A?

P19. Suponha que os nós A e B estejam no mesmo segmento de uma Ethernet de 10 Mbits/s e que o atraso de propagação entre os dois nós seja de 245 tempos de bit. Imagine que A e B enviem quadros ao mesmo tempo, que estes colidam e que, então, A e B escolham valores diferentes de K no algoritmo CSMA/CD. Admitindo que nenhum outro nó esteja ativo, as retransmissões de A e B podem colidir? Para nossa finalidade, é suficiente resolver o seguinte exemplo. Suponha que A e B comecem a transmitir em $t = 0$ tempos de bit. Ambos detectam colisões em $t = 245$ tempos de bit. Suponha que $K_A = 0$ e $K_B = 1$. Em que tempo B programa sua retransmissão? Em que tempo A começa a transmissão? (*Nota*: os nós devem esperar por um canal ocioso após retornar à etapa 2 — veja o protocolo.) Em que tempo o sinal de A chega a B? B se abstém de transmitir em seu tempo programado?

P20. Neste problema, você derivará a eficiência de um protocolo de acesso múltiplo semelhante ao CSMA/CD. Nele, o tempo é segmentado e todos os adaptadores estão sincronizados com os intervalos. Entretanto, diferentemente do *slotted* ALOHA, o comprimento de um intervalo (em segundos) é muito menor do que um tempo de quadro (o tempo para transmitir um quadro). Seja S o comprimento de um intervalo. Suponha que todos os quadros tenham comprimento constante $L = kRS$, sendo R a taxa de transmissão do canal e k um número inteiro grande. Suponha que haja N nós, cada um com um número infinito de quadros para enviar. Admitimos também que $d_{prop} < S$, de modo que todos os nós podem detectar uma colisão antes do final de um intervalo de tempo. O protocolo é o seguinte:

- Se, para determinado intervalo, nenhum nó estiver de posse do canal, todos disputam o canal; em particular, cada nó transmite no intervalo com probabilidade p. Se exatamente um nó transmitir no intervalo, esse nó tomará posse do canal para os $k - 1$ intervalos subsequentes e transmitirá seu quadro inteiro.
- Se algum nó estiver de posse do canal, todos os outros evitarão transmitir até que o nó que está de posse do canal tenha terminado de transmitir seu quadro. Assim que esse nó tiver transmitido seu quadro, todos os nós disputarão o canal.

Note que o canal se alterna entre dois estados: o produtivo, que dura exatamente k intervalos, e o não produtivo, que dura um número aleatório de intervalos. A eficiência do canal é, claramente, a razão $k/(k + x)$, sendo x o número esperado de intervalos consecutivos não produtivos.

a. Para N e p fixos, determine a eficiência desse protocolo.

b. Para N fixo, determine o p que maximiza a eficiência.

c. Usando o p (que é uma função de N) encontrado em (b), determine a eficiência quando N tende ao infinito.

d. Mostre que essa eficiência se aproxima de 1 quando o comprimento do quadro é grande.

P21. Considere a Figura 5.33 no problema P14. Determine os endereços MAC e IP para as interfaces do hospedeiro A, ambos os roteadores e do hospedeiro F. Determine os endereços MAC de origem e destino no quadro que encapsula esse datagrama IP, enquanto o quadro é transmitido *(i)* de A ao roteador esquerdo, *(ii)* do roteador esquerdo ao roteador direito, *(iii)* do roteador direito a F. Determine também os endereços IP de origem e destino no datagrama IP encapsulado no quadro em cada um desses pontos no tempo.

P22. Suponha que o roteador da extremidade esquerda da Figura 5.33 seja substituído por um comutador. Os hospedeiros A, B, C e D e o roteador direito têm uma conexão estrela com esse comutador. Determine os endereços MAC de origem e destino no quadro que encapsula esse datagrama IP enquanto o quadro é transmitido *(i)* de A ao comutador, *(ii)* do comutador ao roteador direito, *(iii)* do roteador direito a F. Determine também os endereços IP de origem e destino no datagrama IP encapsulado pelo quadro em cada um desses pontos no tempo.

P23. Considere a Figura 5.15. Suponha que todos os enlaces têm 100 Mbits/s. Qual é a vazão total máxima agregada que pode ser atingida entre os 9 hospedeiros e 2 servidores nessa rede? Você pode supor que qualquer hospedeiro ou servidor pode enviar a qualquer outro servidor ou hospedeiro. Por quê?

P24. Suponha que três comutadores departamentais na Figura 5.15 são substituídos por *hubs*. Todos os enlaces têm 100 Mbits/s. Agora responda às perguntas feitas no problema P23.

P25. Suponha que *todos* os comutadores na Figura 5.15 sejam substituídos por *hubs*. Todos os enlaces têm 100 Mbits/s. Agora responda às perguntas feitas no problema P23.

P26. Vamos considerar a operação de aprendizagem do comutador no contexto de uma rede em que 6 nós, rotulados de A até F, sejam conectados em estrela a um comutador Ethernet. Suponha que *(i)* B envia um quadro a E, *(ii)* E responde com um quadro a B, *(iii)* A envia um quadro a B, *(iv)* B responde com um quadro a A. A tabela do comutador está inicialmente vazia. Mostre o estado da tabela do comutador antes e depois de cada evento. Para cada um dos eventos, identifique os enlaces em que o quadro transmitido será encaminhado e justifique suas respostas em poucas palavras.

P27. Neste problema, exploraremos o uso de pequenos pacotes de aplicações de voz sobre IP (VoIP). Uma desvantagem de um pacote pequeno é que uma grande parte da largura de banda do enlace é consumida por bytes de cabeçalho. Portanto, suponha que o pacote é formado por P bytes e 5 bytes de cabeçalho.

a. Considere o envio direto de uma fonte de voz codificada digitalmente. Suponha que a fonte esteja codificada a uma taxa constante de 128 Kbits/s. Considere que cada pacote esteja integralmente cheio antes de a fonte enviá-lo para a rede. O tempo exigido para encher um pacote é o **atraso de empacotamento**. Determine, em termos de L, o atraso de empacotamento em milissegundos.

b. Os atrasos de empacotamento maiores do que 20 ms podem causar ecos perceptíveis e desagradáveis. Determine o atraso de empacotamento para L = 1.500 bytes (correspondente, mais ou menos, a um pacote Ethernet de tamanho máximo) e para L = 50 bytes (correspondente a um pacote ATM).

c. Calcule o atraso de armazenagem e repasse em um único comutador para uma taxa de enlace R = 622 Mbits/s para L = 1.500 bytes e L = 50 bytes.

d. Comente as vantagens de usar um pacote de tamanho pequeno.

P28. Considere o único comutador VLAN da Figura 5.25, e suponha que um roteador externo está conectado à porta 1 do comutador. Atribua endereços IP aos hospedeiros EE e CS e à interface do roteador. Relacione as etapas usadas em ambas as camadas, de rede e de enlace, para transferir o datagrama IP ao hospedeiro EE e ao hospedeiro CS. (*Dica:* Leia novamente a discussão sobre a Figura 5.19 no texto.)

P29. Considere a rede MPLS mostrada na Figura 5.29 e suponha que os roteadores R5 e R6 agora são habilitados para MPLS. Imagine que queremos executar engenharia de tráfego de modo que pacotes de R6 destinados a A sejam comutados para A via R6-R4-R3-R1, e pacotes de R5 destinados a A sejam comutados via R5--R4-R2-R1. Mostre as tabelas MPLS em R5 e R6, bem como a tabela modificada em R4, que tornariam isso possível.

P30. Considere a mesma situação do problema anterior, mas suponha que os pacotes de R6 destinados a D estão comutados via R6-R4-R3, enquanto os pacotes de R5 destinados a D sejam comutados via R4-R2-R1-R3. Apresente as tabelas MPLS em todos os roteadores que tornariam isso possível.

P31. Neste problema, você juntará tudo o que aprendeu sobre protocolos de Internet. Suponha que você entre em uma sala, conecte-se à Ethernet e queira fazer o download de uma página. Quais são as etapas de protocolo utilizadas, desde ligar o computador até receber a página? Suponha que não tenha nada no seu DNS ou nos *caches* do seu navegador quando você ligar seu computador. (*Dica:* as etapas incluem o uso de protocolos da Ethernet, DHCP, ARP, DNS, TCP e HTTP.) Indique explicitamente em suas etapas como obter os endereços IP e MAC de um roteador de borda.

P32. Considere a rede do datacenter com topologia hierárquica da Figura 5.30. Suponha agora que haja 80 pares de fluxos, com dez fluxos entre a primeira e a nona estante, dez entre a segunda e a décima estante, e assim por diante. Suponha ainda que todos os enlaces na rede seja de 10 Gbits/s, exceto os enlaces entre os hospedeiros e os comutadores TOR, que são de 1 Gbit/s.

a. Cada fluxo tem a mesma velocidade de dados; determine a velocidade máxima de um fluxo.

b. Para o mesmo padrão de tráfego, determine a velocidade máxima de um fluxo para a topologia altamente interconectada da Figura 5.31.

c. Agora, suponha que haja um padrão de tráfego semelhante, mas envolvendo 20 fluxos em cada hospedeiro e 160 pares de fluxos. Determine as velocidades de fluxo máximas para as duas topologias.

P.33 Considere a rede hierárquica da Figura 5.30 e suponha que o datacenter precise suportar a distribuição de correio eletrônico e vídeo entre outras aplicações. Suponha que quatro estantes de servidores sejam reservadas para correio eletrônico e quatro para vídeo. Para cada aplicação, todas as quatro estantes precisam estar debaixo de um único comutador da camada 2, pois os enlaces entre a camada 2 e a camada 1 não têm largura de banda suficiente para suportar o tráfego dentro da aplicação. Para a aplicação de correio eletrônico, suponha que, durante 99,9% do tempo, só três estantes sejam usadas e que a aplicação de vídeo tenha padrões de uso idênticos.

a. Durante que fração do tempo a aplicação de correio eletrônico precisa usar uma quarta estante? E a aplicação de vídeo?

b. Supondo que o uso de correio eletrônico e o uso de vídeo sejam independentes, durante que fração de tempo (ou, de modo equivalente, qual é a probabilidade de que) as duas aplicações precisam de sua quarta estante?

c. Suponha que seja aceitável para uma aplicação ter um servidor parado por 0,001% do tempo ou menos (causando raros períodos de degradação de desempenho para os usuários). Discuta como a topologia da Figura 5.31 pode ser usada de modo que somente sete estantes sejam coletivamente designadas para as duas aplicações (supondo que a topologia possa suportar todo o tráfego).

WIRESHARK LAB

No site de apoio deste livro você encontrará uma tarefa de laboratório Wireshark, em inglês, que examina a operação do protocolo IEEE 802.3 e o formato do quadro Wireshark. Uma segunda tarefa de laboratório Wireshark examina a sequência dos pacotes usados numa situação de rede doméstica.

ENTREVISTA

Simon S. Lam

Simon S. Lam é professor e regente da cadeira de ciência da computação na Universidade do Texas, em Austin. De 1971 a 1974 trabalhou no ARPA Network Measurement Center na UCLA, com comutação de pacotes por satélite e por rádio. Liderou um grupo de pesquisa que inventou os *sockets* seguros e construiu o primeiro protótipo da camada de *sockets* seguros, em 1993, denominada Programação de Rede Segura (*Secure Network Programming*), pela qual ganhou o ACM Software System Award em 2004. Os pontos de interesse de sua pesquisa são o projeto e a análise de protocolos de rede e serviços de segurança. Graduou-se em engenharia elétrica na Washington State University e é mestre e doutor pela UCLA. Foi eleito à Academia Nacional de Engenharia em 2007.

O que o fez se decidir pela especialização em redes?

Quando cheguei à UCLA, recém-formado, no outono de 1969, minha intenção era estudar teoria de controle. Então assisti às aulas de Leonard Kleinrock sobre teoria das filas e ele me impressionou muito. Durante algum tempo trabalhei em controle adaptativo de sistemas de fila como possível tópico de tese. No início de 1972, Larry Roberts iniciou o projeto ARPAnet Satellite System (mais tarde denominado Packet Satellite). O professor Kleinrock perguntou se eu queria participar do projeto. A primeira coisa que fiz foi introduzir um algoritmo de recuo (*backoff*) simples, mas realista, no protocolo *slotted* ALOHA. Pouco tempo depois, encontrei muitos problemas interessantes para pesquisar, como o problema da instabilidade da ALOHA e a necessidade de recuo, que formariam o núcleo da minha tese.

O senhor teve participação ativa nos primórdios da Internet na década de 1970, desde seus tempos de estudante na UCLA. Como era o panorama naquela época? As pessoas faziam alguma ideia do que a Internet se tornaria?

Na verdade, a atmosfera não era diferente da de outros projetos de construção de sistemas que já vi na indústria e no ambiente acadêmico. A meta inicial determinada para a ARPAnet era bastante modesta, isto é, prover acesso a computadores caros a partir de locais remotos, de modo que mais cientistas pudessem utilizá-los. Contudo, com o início do projeto Packet Satellite em 1972 e do projeto Packet Radio em 1973, a meta da ARPA foi ampliada substancialmente. Em 1973, a ARPA estava montando, ao mesmo tempo, três redes de pacotes diferentes e tornou-se necessário que Vint Cerf e Bob Kahn desenvolvessem uma estratégia de interconexão.

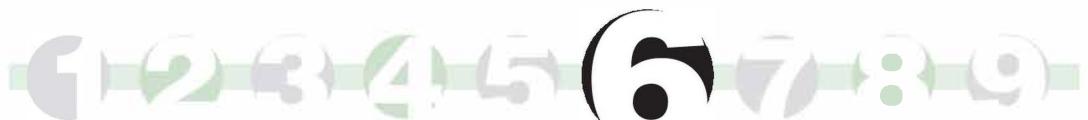
Naquela época, todos esses desenvolvimentos progressivos na área de redes eram vistos (acredito eu) mais como lógicos do que mágicos. Ninguém poderia ter previsto a escalada da Internet e o poder dos computadores pessoais de hoje. Transcorreu uma década antes do aparecimento dos primeiros PCs. Para entender melhor as coisas, a maioria dos estudantes apresentava seus programas de computador em cartões perfurados, para processamento em lote (*batch*). Somente alguns tinham acesso direto a computadores, que em geral eram acomodados em áreas restritas. Os modems eram lentos e ainda raros. No meu tempo de estudante, eu tinha apenas um telefone sobre minha

mesa e usava papel e lápis para fazer a maior parte do meu trabalho.

Em sua opinião, qual é o futuro do campo das redes e da Internet?

No passado, a simplicidade do protocolo IP da Internet era sua maior força para vencer a concorrência e se tornar o padrão na prática do trabalho com redes. Diferentemente de seus concorrentes, como X-25 na década de 1980 e ATM na década de 1990, o IP pode rodar sobre qualquer tecnologia de rede da camada de enlace, porque oferece apenas um serviço de datagrama de melhor esforço. Assim, qualquer rede de pacotes pode se conectar com a Internet.

Hoje, a maior força do IP é na realidade uma deficiência. O IP é como uma camisa de força que confina o desenvolvimento da Internet a direções específicas. Nos últimos anos, muitos pesquisadores redirecionaram seus esforços apenas para as camadas de aplicação. Há também uma grande concentração de pesquisas em redes sem fio e redes ocasionais (*ad hoc*), redes de sensores e redes por satélite. Essas redes podem ser consideradas ou sistemas autônomos ou sistemas da camada de enlace, que podem se desenvolver porque estão fora da camisa de força do IP.

Há muita gente animada com a possibilidade da utilização de sistemas P2P como plataforma para novas aplicações da Internet. Todavia, a utilização de recursos da Internet por sistemas P2P é altamente ineficiente. Uma das minhas preocupações é se a capacidade de transmissão e comutação do núcleo da Internet continuará a crescer mais rapidamente do que a demanda de tráfego na Internet, enquanto ela cresce para interconectar todos os tipos de equipamentos e suportar futuras aplicações habilitadas para P2P. Sem um superprovisionamento substancial de capacidade, garantir a estabilidade da rede na presença de ataques mal-intencionados e de congestionamento seria um desafio significativo.

O crescimento fenomenal da Internet também requer a alocação de novos endereços IP em uma velocidade rápida, para operadores e empresas de rede do mundo inteiro. Se continuarmos neste ritmo, o bloco de endereços IPv4 não alocados se esgotará em alguns anos. Quando isso acontecer, grandes blocos de espaços de endereços só poderão ser alocados de espaços de endereço IPv6. Já que a adoção do IPv6 não está fazendo muito sucesso, pela falta de incentivo para os novos usuários, IPv4 e IPv6 provavelmente coexistirão por muitos anos. A migração bem-sucedida de uma Internet predominantemente IPv4 para uma IPv6 exigirá um grande esforço global.

Qual parte de seu trabalho lhe apresenta mais desafios?

A parte mais desafiadora do meu trabalho como professor é ensinar e motivar *todos* os estudantes que assistem a minhas aulas e *todos* os estudantes de doutorado que supervisiono, e não apenas os mais destacados. Os muito inteligentes e motivados podem exigir um pouco de supervisão, mas não muito mais do que isso. Muitas vezes aprendo mais com esses estudantes do que eles aprendem comigo. Ensinar e motivar os que não se destacam muito é um grande desafio.

Que impactos sobre o aprendizado o senhor acha que a tecnologia terá no futuro?

Com o tempo, quase todo o conhecimento humano estará acessível pela Internet, algo que será a mais poderosa ferramenta de aprendizado. Essa vasta base de conhecimento terá o potencial de nivelar o terreno para estudantes em todo o mundo. Por exemplo, estudantes motivados de qualquer país poderão acessar os melhores sites de aulas, conferências multimídia e material de ensino. Já foi dito que as bibliotecas digitais do IEEE e da ACM aceleraram o desenvolvimento de pesquisadores da ciência da computação na China. Com o tempo, a Internet transcenderá todas as barreiras geográficas ao aprendizado.

C A P Í T U L O

1 2 3 4 5 6 7 8 9

REDES SEM FIO E **REDES MÓVEIS**

No mundo da telefonia pode-se dizer que os quinze últimos anos foram os anos dourados da telefonia celular. O número de assinantes de telefones móveis no mundo inteiro aumentou de 34 milhões em 1993 para quase 5,5 bilhões no final de 2011 e, agora, ultrapassa o número de linhas telefônicas convencionais. As muitas vantagens dos telefones celulares são evidentes para todos — em qualquer lugar, a qualquer hora, acesso desimpedido à rede global de telefonia por meio de um equipamento leve e totalmente portátil. Com o advento de notebooks, palmtops, smartphones e a promessa de acesso desimpedido à Internet global de qualquer lugar, a qualquer hora, será que estamos prestes a assistir a uma explosão semelhante da utilização de dispositivos sem fio para acesso à Internet?

Independentemente do crescimento futuro de equipamentos sem fio para Internet, já ficou claro que redes sem fio e os serviços móveis relacionados que elas possibilitam vieram para ficar. Do ponto de vista de rede, os desafios propostos, em particular nas camadas de enlace e de rede, são tão diferentes dos desafios das redes de computadores cabeadas que é necessário um capítulo inteiro (*este* capítulo) devotado ao estudo de redes sem fio e redes móveis.

Iniciaremos este capítulo com uma discussão sobre usuários móveis, enlaces e redes sem fio e sua relação com as redes maiores (normalmente cabeadas) às quais se conectam. Traçaremos uma distinção entre os desafios propostos pela natureza *sem fio* dos enlaces de comunicação nessas redes e pela *mobilidade* que os enlaces sem fio habilitam. Fazer essa importante distinção — entre sem fio e mobilidade — nos permitirá isolar, identificar e dominar melhor os conceitos fundamentais em cada área. Note que, na realidade, há muitos ambientes de rede nos quais os nós são sem fio, mas não são móveis (por exemplo, redes residenciais sem fio ou redes de escritórios compostas por estações de trabalho estacionárias e monitores de grandes dimensões), e que existem formas limitadas de mobilidade que não requerem enlaces sem fio (por exemplo, um profissional que utiliza um notebook em casa, desliga o equipamento e o leva para seu escritório, onde o liga à rede cabeada da empresa em que trabalha). É claro que muitos dos ambientes sem fio mais interessantes são aqueles em que os usuários são sem fio *e também* móveis — por exemplo, um cenário no qual um usuário móvel (digamos, no banco traseiro de um carro) mantém uma chamada de voz sobre IP (VoIP) e várias conexões TCP ativas enquanto corre pela rodovia a 160 km/h. É nesse ponto, em que o sem fio se cruza com a mobilidade, que encontraremos os desafios técnicos mais interessantes!

Começaremos por ilustrar, primeiro, o cenário no qual consideraremos comunicação e mobilidade sem fio — uma rede na qual usuários sem fio (e possivelmente móveis) estão conectados à infraestrutura da rede maior por um enlace sem fio na borda de rede. Então, consideraremos as características desse enlace sem fio na Seção 6.2. Nessa seção, incluímos uma breve introdução ao acesso múltiplo por divisão de código (*code division*

multiple access — CDMA), um protocolo de acesso ao meio compartilhado que é utilizado com frequência em redes sem fio. Na Seção 6.3, estudaremos com certa profundidade os aspectos da camada de enlace do padrão da LAN sem fio IEEE 802.11 (Wi-Fi); também falaremos um pouco sobre Bluetooth e outras redes pessoais sem fio. Na Seção 6.4 daremos uma visão geral do acesso à Internet por telefone celular, incluindo 3G e as tecnologias celulares emergentes 4G, que fornecem acesso à Internet por voz e em alta velocidade. Na Seção 6.5, voltaremos nossa atenção à mobilidade, focalizando os problemas da localização de um usuário móvel, do roteamento até o usuário móvel e da transferência (*hand-off*) do usuário móvel que passa dinamicamente de um ponto de conexão com a rede para outro. Estudaremos como esses serviços de mobilidade são executados no padrão IP móvel e em GSM nas seções 6.6 e 6.7, respectivamente. Por fim, na Seção 6.8 consideraremos o impacto dos enlaces e da mobilidade sem fio sobre protocolos de camada de transporte e aplicações em rede.

6.1 INTRODUÇÃO

A Figura 6.1 mostra o cenário no qual consideraremos os tópicos de comunicação de dados e mobilidade sem fio. Começaremos mantendo nossa discussão dentro de um contexto geral o suficiente para abranger uma ampla faixa de redes, entre elas LANs sem fio (como a IEEE 802.11) e redes celulares (como uma rede 3G); em outras seções, passaremos então para uma discussão mais detalhada de arquiteturas sem fio específicas. Podemos identificar os seguintes elementos em uma rede sem fio:

- *Hospedeiros sem fio*. Como no caso de redes cabeadas (ou com fio), hospedeiros são os equipamentos de sistemas finais que executam aplicações. Um **hospedeiro sem fio** pode ser um notebook, um palmtop, um smartphone ou um computador de mesa. Os hospedeiros em si podem ser móveis ou não.
- *Enlaces sem fio*. Um hospedeiro se conecta a uma estação-base (definida mais adiante) ou a outro hospedeiro sem fio por meio de um **enlace de comunicação sem fio**. Tecnologias diferentes de enlace sem fio têm taxas de transmissão diversas e podem transmitir a distâncias variadas. A Figura 6.2 mostra duas características fundamentais (área de cobertura e taxa de enlace) dos padrões de enlace sem fio mais populares. (A figura serve apenas para dar uma ideia aproximada dessas características. Por exemplo, alguns desses tipos de redes só estão sendo empregados agora, e algumas taxas de enlace podem aumentar ou diminuir além dos valores mostrados, dependendo da distância, condições do canal e do número de usuários na rede sem fio.) Abordaremos esses padrões mais adiante, na primeira metade deste capítulo; consideraremos também outras características de enlaces sem fio (como suas taxas de erros de bit e as causas desses erros) na Seção 6.2.

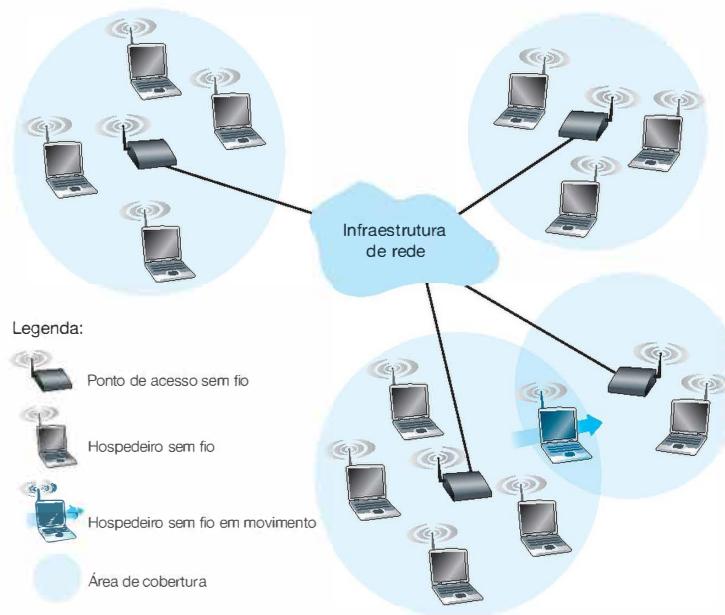
  Na Figura 6.1, enlaces sem fio conectam hospedeiros localizados na borda da rede com a infraestrutura da rede de maior porte. Não podemos nos esquecer de acrescentar que enlaces sem fio às vezes também são utilizados *dentro* de uma rede para conectar roteadores, comutadores e outros equipamentos de rede. Contudo, neste capítulo, focalizaremos a utilização da comunicação sem fio nas bordas da rede, pois é aqui que estão ocorrendo muitos dos desafios técnicos mais interessantes e a maior parte do crescimento.
- *Estação-base*. A **estação-base** é uma parte fundamental da infraestrutura de rede sem fio. Diferentemente dos hospedeiros e enlaces sem fio, uma estação-base não tem nenhuma contraparte óbvia em uma rede cabeada. Uma estação-base é responsável pelo envio e recebimento de dados (por exemplo, pacotes) de e para um hospedeiro sem fio que está associado a ela. Uma estação-base frequentemente será responsável pela coordenação da transmissão de vários hospedeiros sem fio com os quais está associada. Quando dizemos que um hospedeiro sem fio está "associado" a uma estação-base, isso quer dizer que (1) o hospedeiro está dentro do alcance de comunicação sem fio da estação-base e (2) o hospedeiro usa a estação-base para retransmitir dados entre ele (o hospedeiro) e a rede maior. **Torres celulares** em redes celulares e **pontos de acesso** em LANs sem fio 802.11 são exemplos de estações-base.

  Na Figura 6.1, a estação-base está conectada à rede maior (isto é, à Internet, à rede corporativa ou residencial, ou à rede telefônica); portanto, ela funciona como uma retransmissora da camada de enlace entre o hospedeiro sem fio e o resto do mundo com o qual o hospedeiro se comunica.

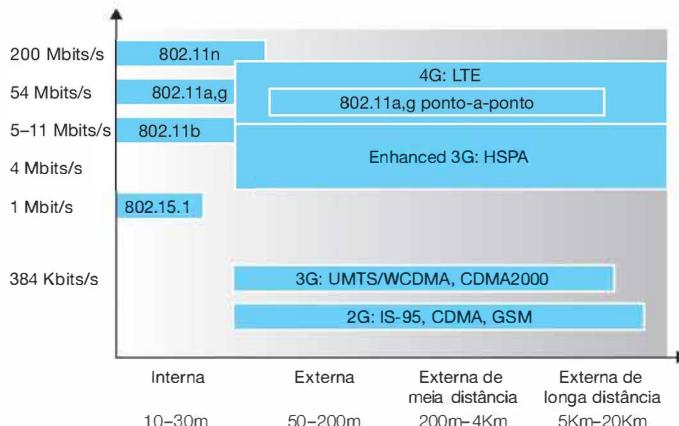
HISTÓRIA

Acesso público Wi-Fi: em breve, em um poste próximo de você?

Pontos de acesso Wi-Fi — locais públicos onde os usuários podem encontrar acesso sem fio 802.11 — estão se tornando cada vez mais comuns em hotéis, aeroportos e cafés ao redor do mundo. A maioria dos *campi* universitários oferece acesso sem fio espalhado por toda a parte, e é difícil encontrar um hotel que não oferece acesso à Internet sem fio.

Durante a última década, diversas cidades projetaram, implantaram e operaram redes Wi-Fi municipais. A visão de oferecer acesso Wi-Fi por toda a parte para a comunidade como um serviço público (semelhante aos postes de luz) — ajudando a eliminar a exclusão digital por meio do acesso à Internet para todos os cidadãos e a promover o desenvolvimento econômico — é tentadora. Muitas cidades do mundo inteiro, incluindo Filadélfia, Toronto, Hong Kong, Minneapolis, Londres e Auckland, anunciaram planos de prover esse acesso sem fio dentro de todo o município, ou já fizeram isso de formas variadas. O objetivo na Filadélfia foi "transformar a Filadélfia no maior ponto de acesso Wi-Fi do país e ajudar a melhorar a educação, eliminar a exclusão digital, aprimorar o desenvolvimento da região e reduzir os custos do governo". O ambicioso programa — um acordo entre a cidade, a Wireless Philadelphia (entidade sem fins lucrativos) e o ISP Earthlink — construiu uma rede operacional de pontos de acesso 802.11b nos braços de poste de iluminação e semáforos, abrangendo 80% da cidade. Porém, questões financeiras e operacionais fizeram a rede ser vendida a um grupo de investidores privados em 2008, que mais tarde a revenderam para a cidade em 2010. Outros centros, como Minneapolis, Toronto, Hong Kong e Auckland, tiveram sucesso com esforços em menor escala.

O fato de que redes 802.11 operam no espectro não licenciado (e por isso podem ser realizadas sem a compra dos caríssimos direitos de uso do espectro) parece torná-las financeiramente atraentes. Porém, os pontos de acesso 802.11 (ver Seção 6.3) possuem alcance muito mais curto que as estações-base de celular 3G (ver Seção 6.4), exigindo um maior número de pontos de acesso para cobrir a mesma região geográfica. Por outro lado, as redes de dados por celular, que oferece acesso à Internet, operam no espectro licenciado. As operadoras de celular pagam bilhões de dólares pelos direitos de acesso ao espectro para suas redes, tornando as redes de dados por celular um negócio lucrativo, em vez de um empreendimento municipal.

Quando hospedeiros estão associados com uma estação-base, em geral diz-se que estão operando em **modo de infraestrutura**, já que todos os serviços tradicionais de rede (por exemplo, atribuição de endereço e roteamento) são fornecidos pela rede com a qual estiverem conectados por meio da estação-base. Em **redes *ad hoc***, hospedeiros sem fio não dispõem de qualquer infraestrutura desse tipo com a qual possam se conectar. Na ausência de tal infraestrutura, os próprios hospedeiros devem prover serviços como roteamento, atribuição de endereço, tradução de endereços semelhante ao DNS e outros.

Quando um hospedeiro móvel se desloca para fora da faixa de alcance de uma estação-base e entra na faixa de outra, ele muda seu ponto de conexão com a rede maior (isto é, muda a estação-base com a qual está associado) — um processo denominado **transferência** (*handoff*). Essa mobilidade dá origem a muitas questões desafiadoras. Se um hospedeiro pode se mover, como descobrir sua localização atual na rede de modo que seja possível lhe encaminhar dados? Como é realizado o endereçamento, visto que um hospedeiro pode estar em um dentre muitos locais possíveis? Se o hospedeiro se movimentar *durante* uma conexão TCP ou ligação telefônica, como os dados serão roteados para que a conexão continue sem interrupção? Essas e muitas (mas muitas!) outras questões fazem das redes sem fio e móveis uma área de pesquisa muito interessante sobre redes.

- *Infraestrutura de rede.* É a rede maior com a qual um hospedeiro sem fio pode querer se comunicar.

Após discutir sobre as "partes" da rede sem fio, observamos que essas partes podem ser combinadas de diversas maneiras diferentes para formar diferentes tipos de redes sem fio. Você pode achar uma taxonomia desses tipos de redes sem fio útil ao ler este capítulo, ou ler/aprender mais sobre redes sem fio além deste livro. No nível mais alto,

FIGURA 6.1 ELEMENTOS DE UMA REDE SEM FIO

podemos classificar as redes sem fio de acordo com dois critérios: *(i)* se um pacote na rede sem fio atravessa exatamente *um salto único sem fio ou múltiplos saltos sem fio*, e *(ii)* se há *infraestrutura* na rede, como uma estação-base:

- *Salto único, com infraestrutura.* Essas redes têm uma estação-base conectada a uma rede cabeada maior (por exemplo, a Internet). Além disso, toda a comunicação é feita entre a estação-base e um hospedeiro sem fio através de um único salto sem fio. As redes 802.11 que você utiliza na sala de aula, na lanchonete ou na biblioteca; e as redes de dados por celular 3G, que aprenderemos em breve, encaixam-se nesta categoria.

FIGURA 6.2 CARACTERÍSTICAS DE ENLACES DE PADRÕES SELECIONADOS DE REDE SEM FIO

- *Salto único, sem infraestrutura.* Nessas redes, não existe estação-base conectada à rede sem fio. Entretanto, como veremos, um dos nós nessa rede de salto único pode coordenar as transmissões dos outros nós. As redes Bluetooth (que serão estudadas na Seção 6.3.6) e as redes 802.11 no modo *ad hoc* são redes de salto único, sem infraestrutura.
- *Múltiplos saltos, com infraestrutura.* Nessas redes, está presente uma estação-base cabeada para as redes maiores. Entretanto, alguns nós sem fio podem ter que restabelecer sua comunicação através de outros nós sem fio para se comunicarem por meio de uma estação-base. Algumas redes de sensores sem fio e as chamadas redes em malha sem fio se encaixam nesta categoria.
- *Múltiplos saltos, sem infraestrutura.* Não existe estação-base nessas redes, e os nós podem ter de restabelecer mensagens entre diversos outros nós para chegar a um destino. Os nós também podem ser móveis, ocorrendo mudança de conectividade entre eles — uma categoria de redes conhecida como redes móveis ***ad hoc*** (MANETs). Se os nós móveis forem veículos, essa rede é denominada rede veicular ***ad hoc*** (VANET). Como você pode imaginar, o desenvolvimento de protocolos para essas redes é desafiador e constitui o assunto de muita pesquisa em andamento.

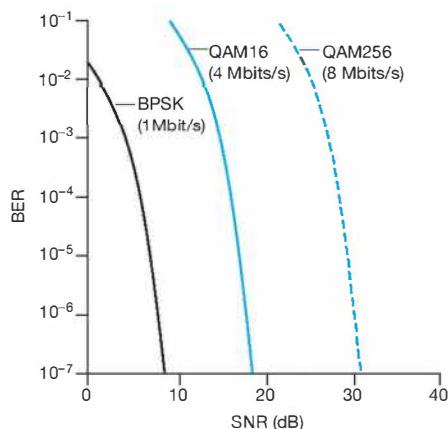
Neste capítulo, vamos nos limitar às redes de salto único e, depois, principalmente às redes baseadas em infraestrutura.

Agora vamos nos aprofundar um pouco mais nos desafios técnicos que surgem em redes sem fio e móveis. Começaremos considerando, em primeiro lugar, o enlace sem fio individual, deixando nossa discussão sobre mobilidade para outra parte deste capítulo.

6.2 CARACTERÍSTICAS DE ENLACES E REDES SEM FIO

Vamos começar considerando uma rede simples cabeada, por exemplo, uma rede residencial, com hospedeiros interconectados por um comutador Ethernet cabeado (ver Seção 5.4). Se substituíssemos a Ethernet cabeada por uma rede 802.11 sem fio, uma interface de rede sem fio substituiria a interface Ethernet cabeada nos hospedeiros e um ponto de acesso substituiria o comutador Ethernet, mas, na camada de rede ou acima dela, praticamente nenhuma mudança seria necessária. Isso sugere que concentremos nossa atenção na camada de enlace ao procurarmos diferenças importantes entre redes com fio e sem fio. Realmente, podemos encontrar várias diferenças importantes entre um enlace com fio e um enlace sem fio:

- *Redução da força do sinal.* Radiações eletromagnéticas são atenuadas quando atravessam algum tipo de matéria (por exemplo, um sinal de rádio ao atravessar uma parede). O sinal se dispersará mesmo ao ar livre, resultando na redução de sua força (às vezes denominada atenuação de percurso) à medida que aumenta a distância entre emissor e receptor.
- *Interferência de outras fontes.* Várias fontes de rádio transmitindo na mesma banda de frequência sofrerão interferência umas das outras. Por exemplo, telefones sem fio de 2,4 GHz e LANs sem fio 802.11b transmitem na mesma banda de frequência. Assim, o usuário de uma LAN sem fio 802.11b que estiver se comunicando por um telefone sem fio de 2,4 GHz pode esperar que nem a rede nem o telefone funcionem particularmente bem. Além da interferência de fontes transmissoras, o ruído eletromagnético presente no ambiente (por exemplo, um motor ou um equipamento de micro-ondas próximo) pode causar interferência.
- *Propagação multivias.* A propagação multivias (ou multicaminhos) ocorre quando partes da onda eletromagnética se refletem em objetos e no solo e tomam caminhos de comprimentos diferentes entre um emissor e um receptor. Isso resulta no embaralhamento do sinal recebido no destinatário. Objetos que se movimentam entre o emissor e o receptor podem fazer com que a propagação multivias mude ao longo do tempo.

Para obter uma discussão detalhada sobre as características, modelos e medidas do canal sem fio, consulte Anderson [1995].

A discussão anterior sugere que erros de bit serão mais comuns em enlaces sem fio do que em enlaces com fio. Por essa razão, talvez não seja nenhuma surpresa que protocolos de enlace sem fio (como o protocolo 802.11 que examinaremos na seção seguinte) empreguem não só poderosos códigos de detecção de erros por CRC, mas também protocolos de transferência de dados confiável em nível de enlace, que retransmitem quadros corrompidos.

Tendo considerado as falhas que podem ocorrer em um canal sem fio, vamos voltar nossa atenção para o hospedeiro que recebe o sinal sem fio. Esse hospedeiro recebe um sinal eletromagnético que é uma combinação de uma forma degradada do sinal original transmitido pelo remetente (degradada pelos efeitos da atenuação e da propagação multivias, discutidas acima, entre outros) e um ruído de fundo no ambiente. A **relação sinal-ruído** (SNR — *signal-to-noise ratio*) é uma medida relativa da potência do sinal recebido (ou seja, a informação sendo transmitida) e o ruído. A SNR costuma ser calculada em unidades de decibéis (dB), uma unidade de medida que, segundo alguns, é utilizada por engenheiros elétricos principalmente para confundir cientistas da computação. A SNR, medida em dB, é vinte vezes a razão do logaritmo de base 10 da amplitude do sinal recebido à amplitude do ruído. Para nossos fins, precisamos saber apenas que uma SNR maior facilita ainda mais para o destinatário extrair o sinal transmitido de um ruído de fundo.

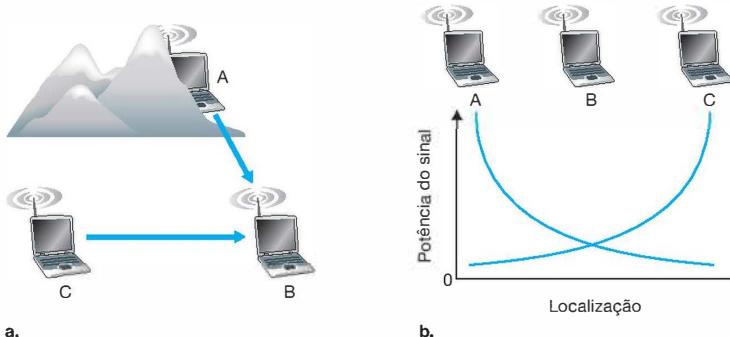
A Figura 6.3 (adaptada de Holland [2001]) mostra a taxa de erro de bits (BER — *bit error rate*) — em termos simples, a probabilidade de um bit transmitido ser recebido com erro no destinatário — *versus* a SNR para três técnicas de modulação diferentes para codificar informações para a transmissão em um canal sem fio idealizado. A teoria da modulação e da codificação, bem como a extração do sinal e a BER, vai além do escopo deste livro (consulte Schwartz [1980] para obter uma discussão sobre esses assuntos). Não obstante, a Figura 6.3 ilustra diversas características da camada física que são importantes para entender os protocolos de comunicação sem fio da camada superior:

- *Para um determinado esquema de modulação, quanto mais alta for a SNR, mais baixa será a BER.* Visto que um remetente consegue aumentar a SNR elevando sua potência de transmissão, ele pode reduzir a probabilidade de um quadro ser recebido com erro diminuindo tal potência. Observe, entretanto, que há um pequeno ganho prático no aumento da potência além de certo patamar, digamos que para diminuir a BER de 10^{-12} para 10^{-13}. Existem também *desvantagens* associadas com o aumento da potência de transmissão: mais energia deve ser gasta pelo remetente (uma consideração importante para usuários móveis, que utilizam bateria), e as transmissões do remetente têm mais probabilidade de interferir nas transmissões de outro remetente (consulte Figura 6.4(b)).
- *Para determinada SNR, uma técnica de modulação com uma taxa de transmissão de bit maior (com erro ou não) terá uma BER maior.* Por exemplo, na Figura 6.3, com uma SNR de 10 dB, a modulação BPSK com uma taxa de transmissão de 1 Mbit/s possui uma BER menor do que 10^{-7}, enquanto para a modulação QAM16 com uma taxa de transmissão de 4 Mbits/s, a BER é 10^{-1}, longe de ser útil na prática. Entretanto, com uma SNR de 20 dB, a modulação QAM16 possui uma taxa de transmissão de 4 Mbits/s e uma BER de 10^{-7}, enquanto a modulação BPSK possui uma taxa de transmissão de apenas 1 Mbit/s e uma BER tão baixa como estar (literalmente) "fora da parada". Se é possível suportar uma BER de 10^{-7}, a taxa de transmissão mais alta apresentada pela modulação QAM16 faria desta a técnica de modulação preferida nesta situação. Tais considerações dão origem à característica final, descrita a seguir.
- *A seleção dinâmica da técnica de modulação da camada física pode ser usada para adaptar a técnica de modulação para condições de canal.* A SNR (e, portanto, a BER) pode mudar, como resultado da mobilidade ou em razão das mudanças no ambiente. A modulação adaptativa e a codificação são usadas em sistemas de dados celulares e nas redes de dados Wi-Fi 802.11 e celular 3G, que estudaremos nas Seções 6.3 e 6.4. Isso permite, por exemplo, a seleção de uma técnica de modulação que ofereça a mais alta taxa de transmissão possível sujeita a uma limitação na BER, para as características de determinado canal.

FIGURA 6.3 TAXA DE ERRO DE BITS, TAXA DE TRANSMISSÃO E SNR

Taxas de erros de bits mais altas e que variam com o tempo não são as únicas diferenças entre um enlace com fio e um enlace sem fio. Lembre-se de que, no caso de enlaces de difusão cabeados, cada nó recebe as transmissões de todos os outros nós. No caso de enlaces sem fio, a situação não é tão simples, conforme mostra a Figura 6.4. Suponha que a estação A esteja transmitindo para a estação B. Suponha também que a estação C esteja transmitindo para a estação B. O denominado **problema do terminal oculto**, obstruções físicas presentes no ambiente (por exemplo, uma montanha ou um prédio), pode impedir que A e C escutem as transmissões um do outro, mesmo que as transmissões de A e C interfiram no destino, B. Isso é mostrado na Figura 6.4(a). Um segundo cenário que resulta em colisões que não são detectadas no receptor é causado pelo **desvanecimento** da força de um sinal à medida que se propaga pelo meio sem fio. A Figura 6.4(b) ilustra o caso em que a localização de A e C é tal que as potências de seus sinais não são suficientes para que eles detectem as transmissões um do outro, mas, mesmo assim, *são* fortes o bastante para interferir uma com a outra na estação B. Como veremos na Seção 6.3, o problema do terminal oculto e o desvanecimento tornam o acesso múltiplo em uma rede sem fio consideravelmente mais complexo do que em uma rede cabeada.

FIGURA 6.4 PROBLEMA DO TERMINAL OCULTO (a) E DO DESVANECIMENTO (b)

6.2.1 CDMA

Lembre-se de que dissemos, no Capítulo 5, que, quando hospedeiros se comunicam por um meio compartilhado, é preciso um protocolo para que os sinais enviados por vários emissores não interfiram nos receptores. No mesmo capítulo descrevemos três classes de protocolos de acesso ao meio: de partição de canal, de acesso aleatório e de revezamento. O acesso múltiplo por divisão de código (*code division multiple access* — CDMA) pertence à família de protocolos de partição de canal. Ele predomina em tecnologias de LAN sem fio e celulares. Por ser tão importante no mundo sem fio, examinaremos o CDMA rapidamente agora, antes de passar para tecnologias específicas de acesso sem fio nas próximas seções.

Com um protocolo CDMA, cada bit que está sendo enviado é codificado pela multiplicação do bit por um sinal (o código) que muda a uma velocidade muito maior (conhecida como **taxa de *chipping***) do que a sequência original de bits de dados. A Figura 6.5 mostra um cenário simples e idealizado de codificação/decodificação CDMA. Suponha que a velocidade com que bits de dados originais cheguem ao codificador CDMA defina a unidade de tempo; isto é, cada bit original de dados a ser transmitido requer um intervalo de tempo de um bit. Seja d_i o valor do bit de dados para o i-ésimo intervalo de bit. Por conveniência do cálculo matemático, representamos o bit de dados com valor 0 por −1. Cada intervalo de bit é ainda subdividido em M mini-intervalos. Na Figura 6.5, $M = 8$, embora, na prática, M seja muito maior. O código CDMA usado pelo remetente consiste em uma sequência de M valores, c_m, $m = 1, \ldots, M$, cada um assumindo um valor de +1 ou −1. No exemplo da Figura 6.5, o código CDMA de M bits que está sendo usado pelo remetente é (1, 1, 1, −1, 1, −1, −1, −1).

Para ilustrar como o CDMA funciona, vamos focalizar o i-ésimo bit de dados, d_i. Para o m-ésimo mini-intervalo do tempo de transmissão de bits de d_i, a saída do codificador CDMA, $Z_{i,m}$, é o valor de d_i multiplicado pelo m-ésimo bit do código CDMA escolhido, c_m:

$$Z_{i,m} = d_i \cdot c_m \tag{6.1}$$

Se o mundo fosse simples e não houvesse remetentes interferindo, o receptor receberia os bits codificados, $Z_{i,m}$, e recuperaria os bits de dados originais, d_i, calculando:

$$d_i = \frac{1}{M} \sum_{m=1}^{M} Z_{i,m} \cdot c_m \tag{6.2}$$

Talvez o leitor queira repassar os detalhes do exemplo da Figura 6.5 para verificar se os bits originais de dados são, de fato, corretamente recuperados no receptor usando a Equação 6.2.

No entanto, o mundo está longe de ser ideal e, como mencionamos antes, o CDMA deve funcionar na presença de remetentes que interferem e que estão codificando e transmitindo seus dados usando um código designado diferente. Mas, como um receptor CDMA pode recuperar bits de dados originais de um remetente quando estes estão sendo embaralhados com bits que estão sendo transmitidos por outros remetentes? O CDMA trabalha na hipótese de que os sinais de bits interferentes sendo transmitidos são aditivos. Isso significa, por exemplo, que, se três remetentes enviam um valor 1 e um quarto envia um valor −1 durante o mesmo mini-intervalo, então o sinal recebido em todos os receptores durante o mini-intervalo é 2 (já que 1 + 1 + 1 − 1 = 2). Na presença de vários remetentes, s calcula suas transmissões codificadas, $Z^s_{i,m}$, exatamente como na Equação 6.1. O valor recebido no receptor durante o m-ésimo mini-intervalo do i-ésimo intervalo de bit, contudo, é agora a *soma* dos bits transmitidos de todos os N remetentes durante o mini-intervalo:

$$Z^*_{i,m} = \sum_{s=1}^{N} Z^s_{i,m}$$

Surpreendentemente, se os códigos dos remetentes forem escolhidos com cuidado, cada receptor pode recuperar os dados enviados por um dado remetente a partir do sinal agregado apenas usando o código do remetente, como na Equação 6.2:

$$d_i = \frac{1}{M}\sum_{m=1}^{M} Z_{i,m}^{*} \cdot c_m \qquad (6.3)$$

A Figura 6.6 ilustra um exemplo de CDMA com dois remetentes. O código CDMA de *M* bits usado pelo remetente que está acima é (1, 1, 1, −1, 1, −1, −1, −1), ao passo que o código CDMA usado pelo que está embaixo é (1, −1, 1, 1, 1,−1, 1, 1). A Figura 6.6 ilustra um receptor recuperando os bits de dados originais do remetente que está acima. Note que o receptor pode extrair os dados do remetente 1, a despeito da transmissão interferente do remetente 2.

Voltando à analogia do coquetel apresentada no Capítulo 5, um protocolo CDMA é semelhante à situação em que os convidados falam vários idiomas; nessa circunstância, os seres humanos até que são bons para manter conversações no idioma que entendem e, ao mesmo tempo, continuar filtrando (rejeitando) outras conversações. Vemos aqui que o CDMA é um protocolo de partição, pois reparte o espaço de código (e não o tempo ou a frequência) e atribui a cada nó uma parcela dedicada do espaço de código.

Nossa discussão do código CDMA aqui é necessariamente breve; na prática, devem ser abordadas inúmeras questões diferentes. Primeiro, para que receptores CDMA consigam extrair o sinal de um emissor qualquer, os códigos CDMA devem ser escolhidos cuidadosamente. Segundo, nossa discussão considerou que as intensidades dos sinais recebidos de vários emissores são as mesmas; na realidade, isso pode ser difícil de conseguir. Existe muita literatura abordando essas e outras questões relativas ao CDMA; veja Pickholtz [1982]; Viterbi [1995], se quiser mais detalhes.

FIGURA 6.5 UM EXEMPLO SIMPLES DE CDMA: CODIFICAÇÃO NO REMETENTE, DECODIFICAÇÃO NO RECEPTOR

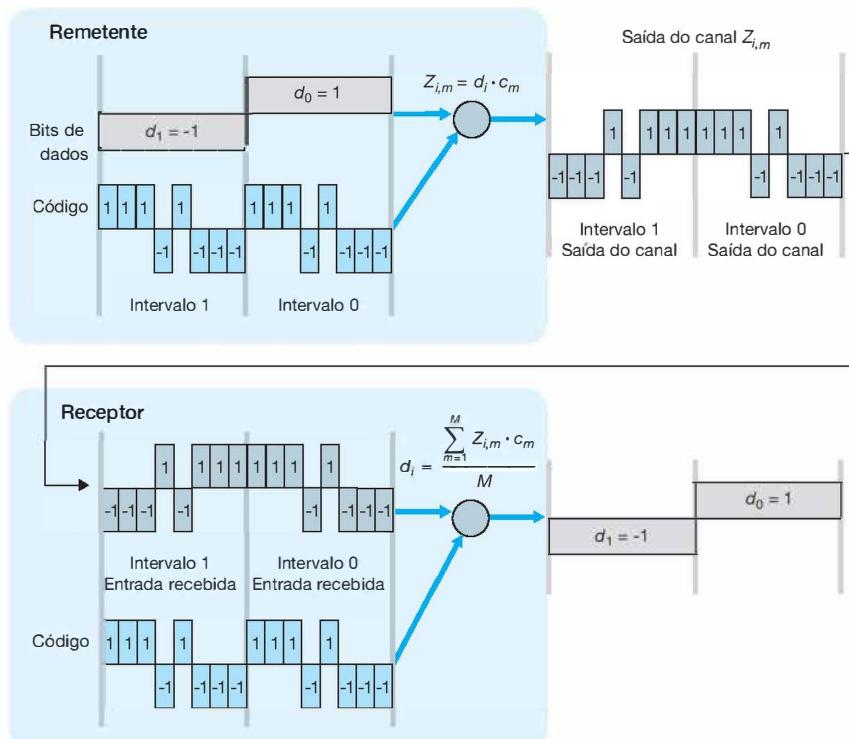
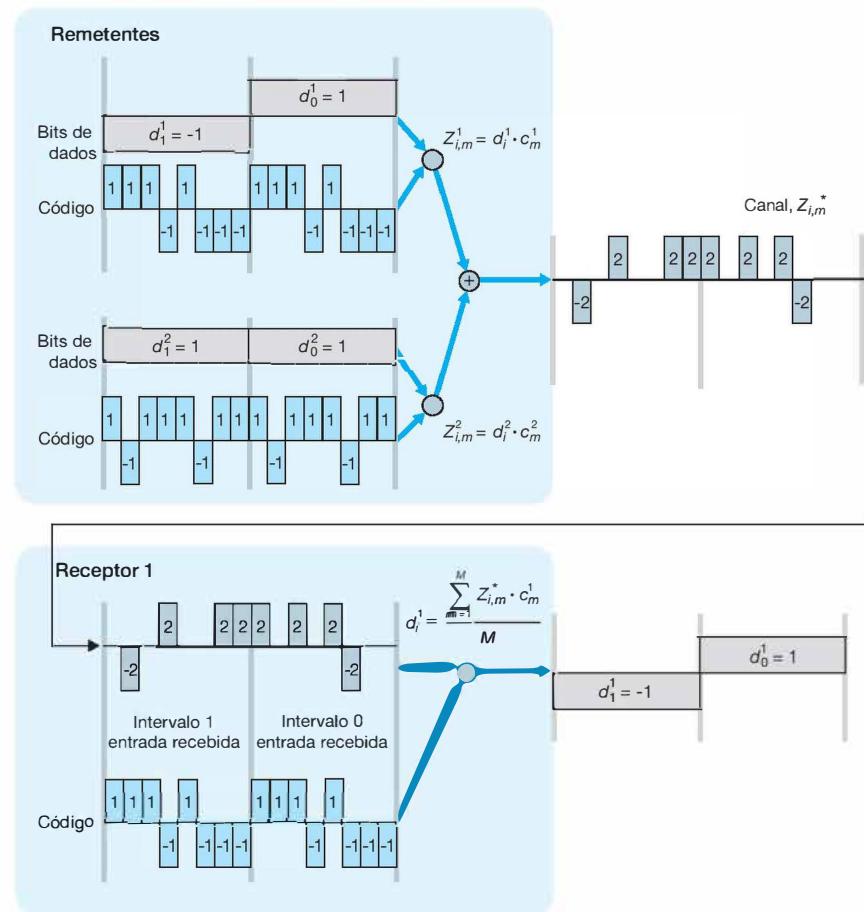



FIGURA 6.6 UM EXEMPLO DE CDMA COM DOIS REMETENTES

6.3 WI-FI: LANS SEM FIO 802.11

Presentes no local de trabalho, em casa, em instituições educacionais, em cafés, aeroportos e esquinas, as LANs sem fio agora são uma das mais importantes tecnologias de rede de acesso na Internet de hoje. Embora muitas tecnologias e padrões para LANs sem fio tenham sido desenvolvidos na década de 1990, uma classe particular de padrões surgiu claramente como a vencedora: a **LAN sem fio IEEE 802.11**, também conhecida como **Wi-Fi**. Nesta seção estudaremos em mais detalhes as LANs sem fio 802.11, examinando a estrutura do quadro 802.11, o protocolo 802.11 de acesso ao meio e a interconexão de LANs 802.11 com LANs Ethernet cabeadas.

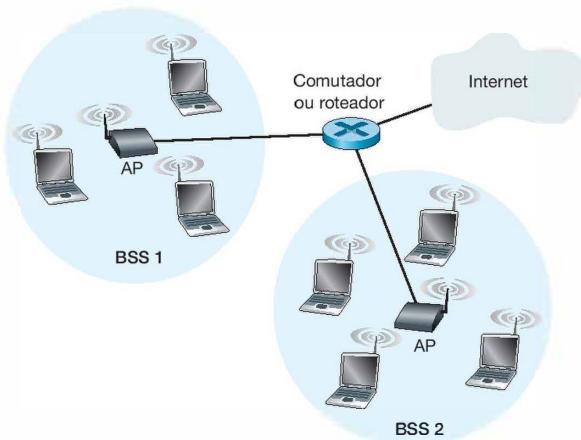
Há diversos padrões 802.11 para tecnologia de LAN sem fio, entre eles 802.11b, 802.11a e 802.11g. A Tabela 6.1 apresenta um resumo das principais características desses padrões. 802.11g é, de longe, a tecnologia mais popular. Estão também disponíveis diversos mecanismos de modos duplo (802.11a/g) e triplo (802.11a/b/g).

Os três padrões 802.11 compartilham muitas características. Todos usam o mesmo protocolo de acesso ao meio, CSMA/CA, que discutiremos em breve. Os três também usam a mesma estrutura de quadro para seus quadros de

camada de enlace. Todos os três padrões têm a capacidade de reduzir sua taxa de transmissão para alcançar distâncias maiores. E todos os três padrões permitem "modo de infraestrutura" e "modo *ad hoc*", como discutiremos em breve. Contudo, conforme mostra a Tabela 6.1, eles apresentam algumas diferenças importantes na camada física.

A LAN sem fio 802.11b tem uma taxa de dados de 11 Mbits/s e opera na faixa de frequência não licenciada de 2,4 a 2,485 GHz, competindo por espectro de frequência com telefones e fornos de micro-ondas de 2,4 GHz. LANs sem fio 802.11a podem funcionar a taxas de bits significativamente mais altas, porém em frequências mais altas. Como operam a uma frequência mais alta, a distância de transmissão dessas LANs é mais curta para determinado nível de potência e elas sofrem mais com a propagação multivias. LANs 802.11g, que operam na mesma faixa de frequência mais baixa das LANs 802.11b e que são compatíveis com a 802.11b (para que se possa atualizar clientes 802.11b de forma incremental), porém com as taxas de transmissão mais altas da 802.11a, devem permitir que os usuários tenham o melhor dos dois mundos.

Um padrão Wi-Fi relativamente novo, 802.11n [IEEE 802.11n, 2012], utiliza antenas de entrada múltipla e saída múltipla (MIMO); ou seja, duas ou mais antenas no lado remetente e duas ou mais antenas no lado destinatário que estão transmitindo/recebendo sinais diferentes [Diggavi, 2004]. Dependendo do esquema de modulação utilizado, é possível alcançar taxas de transmissão de centenas de megabits por segundo com 802.11n.

TABELA 6.1 RESUMO DOS PADRÕES IEEE 802.11

Padrão	Faixa de frequências (EUA)	Taxa de dados
802.11b	2,4–2,485 GHz	até 11 Mbits/s
802.11a	5,1–5,8 GHz	até 54 Mbits/s
802.11g	2,4–2,485 GHz	até 54 Mbits/s

6.3.1 A arquitetura 802.11

A Figura 6.7 ilustra os principais componentes da arquitetura de LAN sem fio 802.11. O bloco de construção fundamental da arquitetura 802.11 é o **conjunto básico de serviço** (*basic service set* — **BSS**). Um BSS contém uma ou mais estações sem fio e uma **estação-base** central, conhecida como um **ponto de acesso** (*access point* — **AP**) na terminologia 802.11. A Figura 6.7 mostra o AP em cada um dos dois BSSs conectando-se a um dispositivo de interconexão (tal como um comutador ou um roteador), que, por sua vez, leva à Internet. Em uma rede residencial típica, há apenas um AP e um roteador (normalmente integrados como uma unidade) que conecta o BSS à Internet.

Como acontece com dispositivos Ethernet, cada estação sem fio 802.11 tem um endereço MAC de 6 bytes que é armazenado no *firmware* do adaptador da estação (isto é, na placa de interface de rede 802.11). Cada AP também tem um endereço MAC para sua interface sem fio. Como na Ethernet, esses endereços MAC são administrados pelo IEEE e são (em teoria) globalmente exclusivos.

Como observamos na Seção 6.1, LANs sem fio que disponibilizam APs em geral são denominadas **LANs sem fio de infraestrutura** e, nesse contexto, "infraestrutura" significa os APs junto com a infraestrutura de Ethernet cabeada que interconecta os APs e um roteador. A Figura 6.8 mostra que estações IEEE 802.11 também podem se agrupar e formar uma rede *ad hoc* — rede sem nenhum controle central e sem nenhuma conexão com o "mundo exterior". Nesse caso, a rede é formada conforme a necessidade, por dispositivos móveis que, por acaso, estão próximos uns dos outros, têm necessidade de se comunicar e não dispõem de infraestrutura de rede no lugar em que se encontram. Uma rede *ad hoc* pode ser formada quando pessoas que portam notebooks se reúnem (por exemplo, em uma sala de conferências, um trem ou um carro) e querem trocar dados na ausência de um AP centralizado. As redes *ad hoc* estão despertando um interesse extraordinário com a contínua proliferação de equipamentos portáteis que podem se comunicar. Porém, nesta seção, concentraremos nossa atenção em LANs sem fio com infraestrutura.

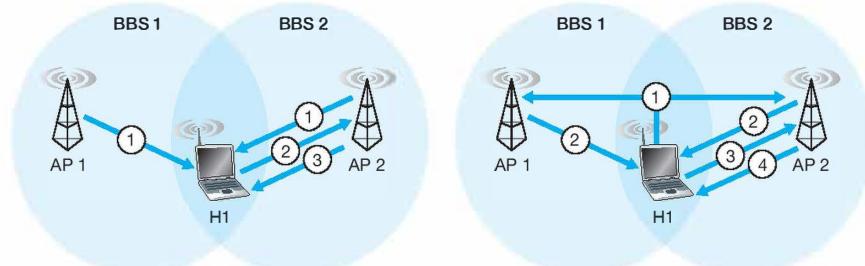
FIGURA 6.7 A ARQUITETURA DE LAN IEEE 802.11

FIGURA 6.8 UMA REDE *AD HOC* IEEE 802.11

Canais e associação

Em 802.11, cada estação sem fio precisa se associar com um AP antes de poder enviar ou receber dados da camada de rede. Embora todos os padrões 802.11 usem associação, discutiremos esse tópico especificamente no contexto da IEEE 802.11b/g.

Ao instalar um AP, um administrador de rede designa ao ponto de acesso um **Identificador de Conjunto de Serviços** (*Service Set Identifier* — SSID) composto de uma ou duas palavras. (O comando "veja redes disponíveis" no Microsoft Windows XP, por exemplo, apresenta uma lista que mostra o SSID de todos os APs ordenado por faixa.) O administrador também deve designar um número de canal ao AP. Para entender números de canal, lembre-se de que as redes 802.11b operam na faixa de frequência de 2,4 GHz a 2,485 GHz. Dentro dessa faixa de 85 MHz, o padrão 802.11 define 11 canais que se sobrepõem em parte. Não há sobreposição entre quaisquer dois canais se, e somente se, eles estiverem separados por quatro ou mais canais. Em particular, o conjunto dos canais 1, 6 e 11 é o único de três canais não sobrepostos. Isso significa que um administrador poderia criar uma LAN sem fio com uma taxa máxima de transmissão agregada de 33 Mbits/s instalando três APs 802.11b na mesma localização física, designando os canais 1, 6 e 11 aos APs e interconectando cada um desses APs com um comutador.

Agora que já entendemos o básico sobre canais 802.11, vamos descrever uma situação interessante (e que não é completamente fora do comum) — uma selva de Wi-Fis. Uma **selva de Wi-Fis** (Wi-Fi *jungle*) é qualquer localização física na qual uma estação sem fio recebe um sinal suficientemente forte de dois ou mais APs. Por exemplo, em muitos cafés da cidade de Nova York, uma estação sem fio pode captar um sinal de diversos APs próximos. Um deles pode ser o AP gerenciado pelo café, enquanto os outros podem estar localizados em apartamentos vizinhos. Cada ponto de acesso provavelmente estaria localizado em uma sub-rede IP diferente e teria sido designado independentemente a um canal.

Agora suponha que você entre nessa selva de Wi-Fis com seu computador portátil, em busca de acesso à Internet sem fio e de um cafezinho. Suponha que há cinco APs na selva de Wi-Fis. Para conseguir acesso à Internet, sua estação sem fio terá de se juntar a exatamente uma das sub-redes e, portanto, precisará se **associar** com exatamente um dos APs. Associar significa que a estação sem fio cria um fio virtual entre ela mesma e o AP. De modo específico, só o AP associado enviará quadros de dados (isto é, quadros contendo dados, tal como um datagrama) a sua estação sem fio e esta enviará quadros de dados à Internet apenas por meio do AP associado. Mas como sua estação sem fio se associa com um determinado AP? E, o que é mais fundamental, como sua estação sem fio sabe quais APs estão dentro da selva, se é que há algum?

O padrão 802.11b requer que um AP envie periodicamente **quadros de sinalização**, cada qual incluindo o SSID e o endereço MAC do AP. Sua estação sem fio, sabendo que os APs estão enviando quadros de sinalização, faz uma varredura dos 11 canais em busca de quadros de sinalização de quaisquer APs que possam estar por lá (alguns dos quais talvez estejam transmitindo no mesmo canal — afinal, estamos na selva!). Ao tomar conhecimento dos APs disponíveis por meio dos quadros de sinalização, você (ou seu hospedeiro sem fio) seleciona um desses pontos de acesso para se associar.

O padrão 802.11 não especifica um algoritmo para selecionar com quais dos APs disponíveis se associar; esse algoritmo é de responsabilidade dos projetistas do *firmware* e do software 802.11 em seu hospedeiro sem fio. Em geral, o hospedeiro escolhe o AP cujo quadro de sinalização é recebido com a intensidade de sinal mais alta. Embora uma intensidade alta do sinal seja algo bom (veja, por exemplo, a Figura 6.3), esta não é a única característica do AP que determinará o desempenho que um hospedeiro recebe. Em particular, é possível que o AP selecionado tenha um sinal forte, mas pode ser sobrecarregado com outros hospedeiros associados (que precisarão compartilhar a largura de banda sem fio naquele AP), enquanto um AP não carregado não é selecionado em razão de um sinal levemente mais fraco. Diversas formas alternativas de escolher os APs foram propostas recentemente [Vasudevan, 2005; Nicholson, 2006; Sudaresan, 2006]. Para obter uma discussão interessante e prática de como a intensidade do sinal é medida, consulte Bardwell [2004].

O processo de varrer canais e ouvir quadros de sinalização é conhecido como **varredura passiva** (veja a Figura 6.9(a)). Um hospedeiro sem fio pode também realizar uma **varredura ativa**, transmitindo um quadro de investigação que será recebido por todos os APs dentro de uma faixa do hospedeiro sem fio, como mostrado na Figura 6.9(b). Os APs respondem ao quadro de requisição de investigação com um quadro de resposta de investigação. O hospedeiro sem fio pode, então, escolher o AP com o qual irá se associar dentre os APs que estão respondendo.

Após selecionar o AP ao qual se associará, o hospedeiro sem fio envia um quadro de solicitação de associação ao AP, e este responde com um quadro de resposta de associação. Observe que essa segunda apresentação de solicitação/resposta é necessária com a varredura ativa, visto que um AP de resposta ao quadro de solicitação de investigação inicial não sabe quais dos (possivelmente muitos) APs de resposta o hospedeiro escolherá para se associar, do mesmo modo que um cliente DHCP pode escolher entre servidores múltiplos DHCP (veja a Figura 4.21). Uma vez associado ao AP, o hospedeiro desejará entrar na sub-rede (no sentido do endereçamento IP da Seção 4.4.2) à qual pertence o AP. Assim, o hospedeiro normalmente enviará uma mensagem de descoberta DHCP (veja a Figura 4.21) à sub-rede por meio de um AP a fim de obter um endereço IP na sub-rede. Logo que o endereço é obtido, o resto do mundo, então, vê esse hospedeiro apenas como outro hospedeiro com um endereço IP naquela sub-rede.

FIGURA 6.9 VARREDURA PASSIVA E ATIVA PARA PONTOS DE ACESSO

a. Varredura passiva
1. Quadros de sinalização enviados dos Aplicações
2. Quadro de Solicitação de Associação enviado: H1 para AP selecionado
3. Quadro de Resposta de Associação enviado: AP selecionado para H1

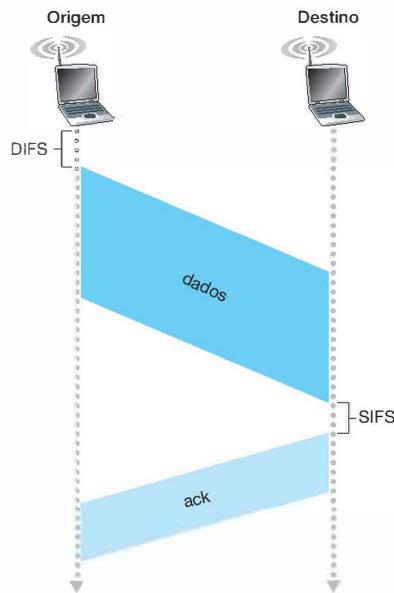
a. Varredura ativa
1. Difusão do quadro de Solicitação de Investigação de H1
2. Quadro de Resposta de Investigações enviado das Aplicações
3. Quadro de Solicitação de Associação enviado: H1 para AP selecionado
4. Quadro de Resposta de Associação enviado: AP selecionado para H1

Para criar uma associação com um determinado AP, a estação sem fio talvez tenha de se autenticar perante o AP. LANs sem fio 802.11 dispõem de várias alternativas para autenticação e acesso. Uma abordagem, usada por muitas empresas, é permitir o acesso a uma rede sem fio com base no endereço MAC de uma estação. Uma segunda abordagem, usada por muitos cafés Internet, emprega nomes de usuários e senhas. Em ambos os casos, o AP em geral se comunica com um servidor de autenticação usando um protocolo como o RADIUS [RFC 2865] ou o DIAMETER [RFC 3588]. Separar o servidor de autenticação do AP permite que um servidor de autenticação atenda a muitos APs, centralizando as decisões de autenticação e acesso (quase sempre delicadas) em um único servidor e mantendo baixos os custos e a complexidade do AP. Veremos, na Seção 8.8, que o novo protocolo IEEE 802.11i, que define aspectos de segurança da família de protocolos 802.11, adota exatamente essa técnica.

6.3.2 O protocolo MAC 802.11

Uma vez associada com um AP, uma estação sem fio pode começar a enviar e receber quadros de dados de e para o ponto de acesso. Porém, como várias estações podem querer transmitir quadros de dados ao mesmo tempo sobre o mesmo canal, é preciso um protocolo de acesso múltiplo para coordenar as transmissões. Aqui, **estação** significa uma estação sem fio ou um AP. Como discutimos no Capítulo 5 e na Seção 6.2.1, em termos gerais, há três classes de protocolos de acesso múltiplo: partição de canal (incluindo CDMA), acesso aleatório e revezamento. Inspirados pelo enorme sucesso da Ethernet e seu protocolo de acesso aleatório, os projetistas do 802.11 escolheram um protocolo de acesso aleatório para as LANs sem fio 802.11. Esse protocolo de acesso aleatório é denominado **CSMA com prevenção de colisão** ou, mais sucintamente, **CSMA/CA**. Do mesmo modo que o CSMA/CD da Ethernet, o "CSMA" de CSMA/CA quer dizer "acesso múltiplo por detecção de portadora", o que significa que cada estação sonda o canal antes de transmitir e abstém-se de transmitir quando percebe que o canal está ocupado. Embora tanto a Ethernet quanto o 802.11 usem acesso aleatório por detecção de portadora, os dois protocolos MAC apresentam diferenças importantes. Primeiro, em vez de usar detecção de colisão, o 802.11 usa técnicas de prevenção de colisão. Segundo, por causa das taxas relativamente altas de erros de bits em canais sem fio, o 802.11 (ao contrário da Ethernet) usa um esquema de reconhecimento/retransmissão (ARQ) de camada de enlace. Mais adiante descreveremos os esquemas usados pelo 802.11 para prevenção de colisão e reconhecimento na camada de enlace.

Lembre-se de que, nas seções 5.3.2 e 5.4.2, dissemos que, com o algoritmo de detecção de colisão, uma estação Ethernet ouve o canal à medida que transmite. Se, enquanto estiver transmitindo, a estação detectar que alguma outra estação também está, ela abortará sua transmissão e tentará novamente após uma pequena unidade de tempo aleatória. Ao contrário do protocolo Ethernet 802.3, o protocolo MAC 802.11 *não* implementa detecção de colisão. Isso se deve a duas razões importantes:

- A capacidade de detectar colisões exige as capacidades de enviar (o próprio sinal da estação) e de receber (para determinar se alguma outra estação está transmitindo) ao mesmo tempo. Como a potência do sinal recebido em geral é muito pequena em comparação com a potência do sinal transmitido no adaptador 802.11, é caro construir um hardware que possa detectar colisões.
- Mais importante, mesmo que o adaptador pudesse transmitir e ouvir ao mesmo tempo (e, presumivelmente, abortar transmissões quando percebesse um canal ocupado), ainda assim ele não seria capaz de detectar todas as colisões, devido ao problema do terminal escondido e do desvanecimento, como discutimos na Seção 6.2.

Como LANs 802.11 sem fio não usam detecção de colisão, uma vez que uma estação comece a transmitir um quadro, *ela o transmite integralmente*; isto é, tão logo uma estação inicie, não há volta. Como é de se esperar, transmitir quadros inteiros (em particular os longos) quando existe grande possibilidade de colisão pode degradar significativamente o desempenho de um protocolo de acesso múltiplo. Para reduzir a probabilidade de colisões, o 802.11 emprega diversas técnicas de prevenção de colisão, que discutiremos em breve.

Antes de considerar prevenção de colisão, contudo, primeiro devemos examinar o esquema de **reconhecimento na camada de enlace** do 802.11. Lembre-se de que dissemos, na Seção 6.2, que, quando uma estação em uma LAN sem fio envia um quadro, este talvez não chegue intacto à estação de destino, por diversos motivos. Para lidar com essa probabilidade não desprezível de falha, o protocolo MAC 802.11 usa reconhecimentos de camada de enlace. Como ilustrado na Figura 6.10, quando a estação de destino recebe um quadro que passou na verificação de CRC, ela espera um curto período de tempo, conhecido como **Espaçamento Curto Interquadros** (*Short Inter-Frame Spacing* — SIFS), e então devolve um quadro de reconhecimento. Se a estação transmissora não receber um reconhecimento em dado período de tempo, ela admitirá que ocorreu um erro e retransmitirá o quadro usando de novo o protocolo CSMA/CA para acessar o canal. Se a estação transmissora não receber um reconhecimento após certo número fixo de retransmissões, desistirá e descartará o quadro.

Agora que já discutimos como o 802.11 usa reconhecimentos da camada de enlace, estamos prontos para descrever o protocolo CSMA/CA 802.11. Suponha que uma estação (pode ser uma estação sem fio ou um AP) tenha um quadro para transmitir.

1. Se inicialmente a estação perceber que o canal está ocioso, ela transmitirá seu quadro após um curto período de tempo conhecido como **Espaçamento Interquadros Distribuído** (*Distributed Inter-Frame Space* — DIFS); ver Figura 6.10.
2. Caso contrário, a estação escolherá um valor aleatório de recuo usando o recuo exponencial binário (conforme encontramos na Seção 5.3.2) e fará a contagem regressiva a partir desse valor quando perceber que o canal está ocioso. Se a estação perceber que o canal está ocupado, o valor do contador permanecerá congelado.
3. Quando o contador chegar a zero (note que isso pode ocorrer somente quando a estação percebe que o canal está ocioso), a estação transmitirá o quadro inteiro e então ficará esperando um reconhecimento.
4. Se receber um reconhecimento, a estação transmissora saberá que o quadro foi corretamente recebido na estação de destino. Se a estação tiver outro quadro para transmitir, iniciará o protocolo CSMA/CA na etapa 2. Se não receber um reconhecimento, a estação entrará de novo na fase de recuo na etapa 2 e escolherá um valor aleatório em um intervalo maior.

Lembre-se de que, no protocolo de acesso múltiplo CSMA/CD (Seção 5.5.2), uma estação começa a transmitir tão logo percebe que o canal está ocioso. Com o CSMA/CA, entretanto, a estação priva-se de transmitir

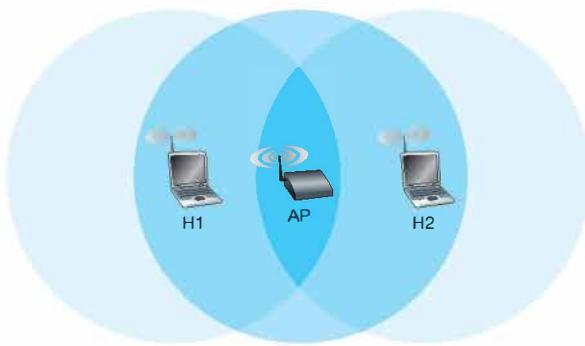



FIGURA 6.10 802.11 USA RECONHECIMENTOS DA CAMADA DE ENLACE

enquanto realiza a contagem regressiva, mesmo quando percebe que o canal está ocioso. Por que o CSMA/CD e o CDMA/CA adotam essas abordagens diferentes aqui?

Para responder a essa pergunta, vamos considerar um cenário com duas estações em que cada uma tem um quadro a transmitir, mas nenhuma transmite imediatamente porque percebe que uma terceira estação já está transmitindo. Com o CSMA/CD da Ethernet, cada uma das duas estações transmitiria tão logo detectasse que a terceira estação terminou de transmitir. Isso causaria uma colisão, o que não é um problema sério em CSMA/CD, já que ambas as estações abortariam suas transmissões e assim evitariam a transmissão inútil do restante dos seus quadros. Entretanto, com 802.11 a situação é bem diferente. Como o 802.11 não detecta uma colisão nem aborta transmissão, um quadro que sofra uma colisão será transmitido integralmente. Assim, a meta do 802.11 é evitar colisões sempre que possível. Com esse protocolo, se duas estações perceberem que o canal está ocupado, ambas entrarão imediatamente em *backoff* aleatório e, esperamos, escolherão valores diferentes de *backoff*. Se esses valores forem, de fato, diferentes, assim que o canal ficar ocioso, uma das duas começará a transmitir antes da outra e (se as duas não estiverem ocultas uma da outra) a "estação perdedora" ouvirá o sinal da "estação vencedora", interromperá seu contador e não transmitirá até que a estação vencedora tenha concluído sua transmissão. Desse modo é evitada uma colisão dispendiosa. É claro que ainda podem ocorrer colisões com 802.11 nesse cenário: as duas estações podem estar ocultas uma da outra ou podem escolher valores de *backoff* aleatório próximos o bastante para que a transmissão da estação que se inicia primeiro tenha ainda de atingir a segunda. Lembre-se de que já vimos esse problema antes em nossa discussão sobre algoritmos de acesso aleatório no contexto da Figura 5.12.

Tratando de terminais ocultos: RTS e CTS

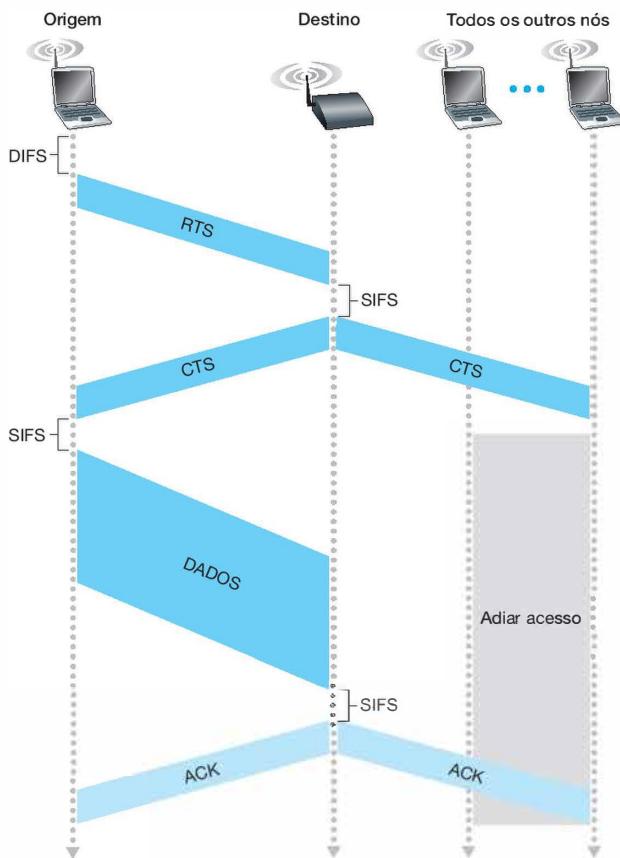
O protocolo 802.11 MAC também inclui um esquema de reserva inteligente (mas opcional) que ajuda a evitar colisões mesmo na presença de terminais ocultos. Vamos estudar esse esquema no contexto da Figura 6.11, que mostra duas estações sem fio e um ponto de acesso. Ambas as estações estão dentro da faixa do AP (cuja área de cobertura é representada por um círculo sombreado) e ambas se associaram com o AP. Contudo, pelo desvanecimento, as faixas

FIGURA 6.11 EXEMPLO DE TERMINAL OCULTO: H1 ESTÁ OCULTO DE H2, E VICE-VERSA

H1
AP
H2

de sinal de estações sem fio estão limitadas ao interior dos círculos sombreados mostrados na Figura 6.11. Assim, cada uma das estações está oculta da outra, embora nenhuma esteja oculta do AP.

Agora vamos considerar por que terminais ocultos podem ser problemáticos. Suponha que a estação H1 esteja transmitindo um quadro e, a meio caminho da transmissão, a estação H2 queira enviar um quadro para o AP. O H2, que não está ouvindo a transmissão de H1, primeiro esperará um intervalo DIFS para, então, transmitir o quadro, resultando em uma colisão. Por conseguinte, o canal será desperdiçado durante todo o período da transmissão de H1, bem como durante a transmissão de H2.

Para evitar esse problema, o protocolo IEEE 802.11 permite que uma estação utilize um quadro de controle **RTS** (*Request to Send* — solicitação de envio) curto e um quadro de controle **CTS** (*Clear to Send* — pronto para envio) curto para *reservar* acesso ao canal. Quando um remetente quer enviar um quadro DATA, ele pode enviar primeiro um quadro RTS ao AP indicando o tempo total requerido para transmitir o quadro DATA e o quadro de reconhecimento (ACK). Quando o AP recebe o quadro RTS, responde fazendo a transmissão por difusão de um quadro CTS. Esse quadro CTS tem duas finalidades: dá ao remetente uma permissão explícita para enviar e também instrui as outras estações a não enviar durante o tempo reservado.

Assim, na Figura 6.12, antes de transmitir um quadro DATA, H1 primeiro faz uma transmissão por difusão de um quadro RTS, que é ouvida por todas as estações que estiverem dentro do seu círculo de alcance, incluindo o AP. O AP então responde com um quadro CTS, que é ouvido por todas as estações dentro de sua faixa de alcance, incluindo H1 e H2. Como ouviu o CTS, a estação H2 deixa de transmitir durante o tempo especificado no quadro CTS. Os quadros RTS, CTS, DATA e ACK são mostrados na Figura 6.12.

A utilização dos quadros RTS e CTS pode melhorar o desempenho de dois modos importantes:

- O problema da estação oculta é atenuado, visto que um quadro DATA longo é transmitido apenas após o canal ter sido reservado.
- Como os quadros RTS e CTS são curtos, uma colisão que envolva um quadro RTS ou CTS terá apenas a duração dos quadros RTS ou CTS curtos. Desde que os quadros RTS e CTS sejam corretamente transmitidos, os quadros DATA e ACK subsequentes deverão ser transmitidos sem colisões.

Aconselhamos o leitor a verificar o applet 802.11 no site deste livro. Esse applet interativo ilustra o protocolo CSMA/CA, incluindo a sequência de troca RTS/CTS.

Embora a troca RTS/CTS ajude a reduzir colisões, também introduz atraso e consome recursos do canal. Por essa razão, a troca RTS/CTS é utilizada (quando utilizada) apenas para reservar o canal para a transmissão de um quadro DATA longo. Na prática, cada estação sem fio pode estabelecer um patamar RTS tal que a sequência RTS/CTS seja utilizada somente quando o quadro for mais longo do que o patamar. Para muitas estações sem fio, o valor *default* do patamar RTS é maior do que o comprimento máximo do quadro, de modo que a sequência RTS/CTS é omitida para todos os quadros DATA enviados.

FIGURA 6.12 PREVENÇÃO DE COLISÃO USANDO OS QUADROS RTS E CTS

Usando o 802.11 como enlace ponto a ponto

Até aqui nossa discussão focalizou a utilização do 802.11 em um cenário de múltiplo acesso. Devemos mencionar que, se dois nós tiverem, cada um, uma antena direcional, eles poderão dirigir suas antenas um para o outro e executar o protocolo 802.11 sobre o que é, essencialmente, um enlace ponto-a-ponto. Dado o baixo custo comercial do hardware 802.11, a utilização de antenas direcionais e uma maior potência de transmissão permitem que o 802.11 seja utilizado como um meio barato de prover conexões sem fio ponto-a-ponto por dezenas de quilômetros. Raman [2007] descreve uma dessas redes sem fio multissaltos que funciona nas planícies rurais do rio Ganges, na Índia, e que contém enlaces 802.11 ponto a ponto.

6.3.3 O quadro IEEE 802.11

Embora o quadro 802.11 tenha muitas semelhanças com um quadro Ethernet, ele também contém vários campos que são específicos para sua utilização para enlaces sem fio. O quadro 802.11 é mostrado na Figura 6.13.

FIGURA 6.13 O QUADRO 802.11

Quadro (os números indicam o comprimento do campo em bytes):

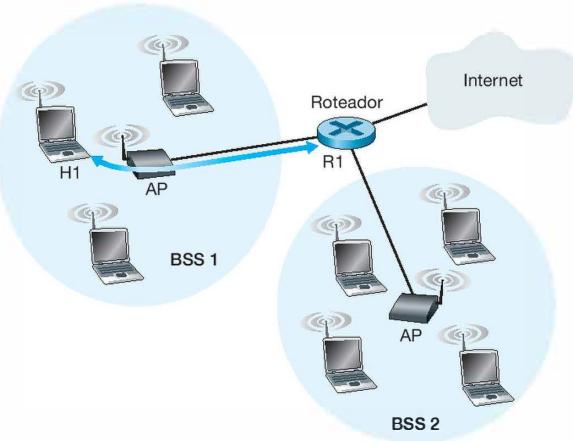
2	2	6	6	6	2	6	0-2312	4
Controle de quadro	Duração	Endereço 1	Endereço 2	Endereço 3	Controle de sequência	Endereço 4	Carga útil	CRC

Detalhamento do campo de controle do quadro (os números indicam o comprimento do campo em bits):

2	2	4	1	1	1	1	1	1	1	1
Versão do protocolo	Tipo	Subtipo	Para o AP	Do AP	Mais frag	Nova tentativa	Ger. de energia	Mais dados	WEP	Reser-vado

Os números acima de cada campo no quadro representam os comprimentos dos campos em *bytes*; os números acima de cada subcampo no campo de controle do quadro representam os comprimentos dos subcampos em *bits*. Agora vamos examinar os campos no quadro, bem como alguns dos subcampos mais importantes no campo de controle do quadro.

Campos de carga útil e de CRC

No coração do quadro está a carga útil, que consiste, tipicamente, em um datagrama IP ou em um pacote ARP. Embora o comprimento permitido do campo seja 2.312 bytes, em geral ele é menor do que 1.500 bytes, contendo um datagrama IP ou um pacote ARP. Como um quadro Ethernet, um quadro 802.11 inclui uma verificação de redundância cíclica (CRC), de modo que o receptor possa detectar erros de bits no quadro recebido. Como já vimos, erros de bits são muito mais comuns em LANs sem fio do que em LANs cabeadas, portanto, aqui, CRC é ainda mais útil.

Campos de endereço

Talvez a diferença mais marcante no quadro 802.11 é que ele tem *quatro* campos de endereço e cada um pode conter um endereço MAC de 6 bytes. Mas por que quatro campos de endereço? Um campo de origem MAC e um campo de destino MAC não são suficientes como são na Ethernet? Acontece que aqueles três campos de endereço são necessários para finalidades de interconexão em rede — especificamente, para mover o datagrama de camada de enlace de uma estação sem fio, passando por um AP, até uma interface de roteador. O quarto campo de endereço é usado quando APs encaminham quadros uns aos outros em modo *ad hoc*. Visto que estamos considerando apenas redes de infraestrutura, vamos concentrar nossa atenção nos três primeiros campos de endereço. O padrão 802.11 define esses campos da seguinte forma:

- Endereço 2 é o endereço MAC da estação que transmite o quadro. Assim, se uma estação sem fio transmitir o quadro, o endereço MAC daquela estação será inserido no campo de endereço 2. De modo semelhante, se um AP transmitir o quadro, o endereço MAC do AP será inserido no campo de endereço 2.
- Endereço 1 é o endereço MAC da estação sem fio que deve receber o quadro. Assim, se uma estação móvel sem fio transmitir o quadro, o endereço 1 conterá o endereço MAC do AP de destino. De modo semelhante, se um AP transmitir o quadro, o endereço 1 conterá o endereço MAC da estação sem fio de destino.
- Para entender o endereço 3, lembre-se de que o BSS (que consiste no AP e estações sem fio) faz parte de uma sub-rede, e que esta se conecta com outras sub-redes, por meio de alguma interface de roteador. O endereço 3 contém o endereço MAC dessa interface de roteador.

Para compreender melhor a finalidade do endereço 3, vamos examinar um exemplo de interconexão em rede no contexto da Figura 6.14. Nessa figura há dois APs, cada um responsável por certo número de estações sem fio. Cada AP tem uma conexão direta com um roteador que, por sua vez, se liga com a Internet global.

FIGURA 6.14 A UTILIZAÇÃO DE CAMPOS DE ENDEREÇO EM QUADROS 802.11: MOVENDO UM QUADRO ENTRE H1 E R1

Devemos ter sempre em mente que um AP é um dispositivo da camada de enlace e, portanto, não "fala" IP nem entende endereços IP. Agora, considere mover um datagrama da interface de roteador R1 até a estação sem fio H1. O roteador não está ciente de que há um AP entre ele e H1; do ponto de vista do roteador, H1 é apenas um hospedeiro em uma das sub-redes às quais ele (o roteador) está conectado.

- O roteador, que conhece o endereço IP de H1 (pelo endereço de destino do datagrama), utiliza ARP para determinar o endereço MAC de H1, exatamente como aconteceria em uma LAN Ethernet comum. Após obter o endereço MAC de H1, a interface do roteador R1 encapsula o datagrama em um quadro Ethernet. O campo de endereço de origem desse quadro contém o endereço MAC de R1 e o campo de endereço de destino contém o endereço MAC de H1.
- Quando o quadro Ethernet chega ao AP, este converte o quadro Ethernet 802.3 para um quadro 802.11 antes de transmiti-lo para o canal sem fio. O AP preenche o endereço 1 e o endereço 2 com o endereço MAC de H1 e seu próprio endereço MAC, respectivamente, como descrito. Para o endereço 3, o AP insere o endereço MAC de R1. Dessa maneira, H1 pode determinar (a partir do endereço 3) o endereço MAC da interface de roteador que enviou o datagrama para a sub-rede.

Agora considere o que acontece quando a estação sem fio H1 responde movendo um datagrama de H1 para R1.

- H1 cria um quadro 802.11, preenchendo os campos de endereço 1 e 2 com o endereço MAC do AP e com o endereço MAC de H1, respectivamente, como descrito. Para o endereço 3, H1 insere o endereço MAC de R1.
- Quando recebe o quadro 802.11, o AP o converte para um quadro Ethernet. O campo de endereço de origem para esse quadro é o endereço MAC de H1 e o campo de endereço de destino é o endereço MAC de R1. Assim, o endereço 3 permite que o AP determine o endereço de destino MAC apropriado ao construir o quadro Ethernet.

Em resumo, o endereço 3 desempenha um papel crucial na interconexão do BSS com uma LAN cabeada.

Campos de número de sequência, duração e controle de quadro

Lembre-se de que, em 802.11, sempre que uma estação recebe corretamente um quadro de outra, devolve um reconhecimento. Como reconhecimentos podem ser perdidos, a estação emissora pode enviar várias cópias

de determinado quadro. Como vimos em nossa discussão sobre o protocolo rdt2.1 (Seção 3.4.1), a utilização de números de sequência permite que o receptor distinga entre um quadro recém-transmitido e a retransmissão de um quadro anterior. Assim, o campo de número de sequência no quadro 802.11 cumpre aqui, na camada de enlace, exatamente a mesma finalidade que cumpria na camada de transporte do Capítulo 3.

Não se esqueça de que o protocolo 802.11 permite que uma estação transmissora reserve o canal durante um período que inclui o tempo para transmitir seu quadro de dados e o tempo para transmitir um reconhecimento. Esse valor de duração é incluído no campo de duração do quadro (tanto para quadros de dados quanto para os quadros RTS e CTS).

Como mostrado na Figura 6.13, o campo de controle de quadro inclui muitos subcampos. Diremos apenas umas poucas palavras sobre alguns dos mais importantes; se o leitor quiser uma discussão mais completa, aconselhamos que consulte a especificação 802.11 [Held, 2001; Crow, 1997; IEEE 802.11, 1999]. Os campos *tipo* e *subtipo* são usados para distinguir os quadros de associação, RTS, CTS, ACK e de dados. Os campos *de* e *para* são usados para definir os significados dos diferentes campos de endereço. (Esses significados mudam dependendo da utilização dos modos *ad hoc* ou de infraestrutura e, no caso do modo de infraestrutura, mudam dependendo de o emissor do quadro ser uma estação sem fio ou um AP.) Finalmente, o campo WEP (*Wireless Equivalent Privacy*) indica se está sendo ou não utilizada criptografia. (A WEP é discutida no Capítulo 8.)

6.3.4 Mobilidade na mesma sub-rede IP

Para ampliar a faixa física de uma LAN sem fio, empresas e universidades frequentemente distribuirão vários BSSs dentro da mesma sub-rede IP. Isso, claro, levanta a questão da mobilidade entre os BSSs — como estações sem fio passam imperceptivelmente de um BSS para outro enquanto mantêm sessões TCP em curso? Como veremos nesta subseção, a mobilidade pode ser manipulada de uma maneira relativamente direta quando os BSSs são parte de uma sub-rede. Quando estações se movimentam entre sub-redes, são necessários protocolos de gerenciamento de mobilidade mais sofisticados, tais como os que estudaremos nas seções 6.5 e 6.6.

Agora vamos examinar um exemplo específico de mobilidade entre BSSs na mesma sub-rede. A Figura 6.15 mostra dois BSSs interconectados e um hospedeiro, H1, que se move entre BSS1 e BSS2. Como nesse exemplo o dispositivo de interconexão entre os dois BSSs *não* é um roteador, todas as estações nos dois BSSs, incluindo os APs, pertencem à mesma sub-rede IP. Assim, quando H1 se move de BSS1 para BSS2, ele pode manter seu endereço IP e todas as suas conexões TCP em curso. Se o dispositivo de interconexão fosse um roteador, então H1 teria de obter um novo endereço IP na sub-rede na qual estava percorrendo. Essa mudança de endereço interromperia (e, em consequência, finalizaria) qualquer conexão TCP em curso em H1. Na Seção 6.6, veremos como um protocolo de mobilidade da camada de rede, como o IP móvel, pode ser usado para evitar esse problema.

Mas o que acontece especificamente quando H1 passa de BSS1 para BSS2? À medida que se afasta de AP1, o H1 detecta um enfraquecimento do sinal de AP1 e começa a fazer uma varredura em busca de um sinal mais forte. H1 recebe quadros de sinalização de AP2 (que em muitos ambientes empresariais e universitários terão o mesmo SSID do AP1). Então, H1 se desassocia de AP1 e se associa com AP2, mantendo, ao mesmo tempo, seu endereço IP e suas sessões TCP em curso.

Isso resolve o problema de transferência do ponto de vista do hospedeiro e do AP. Mas e quanto ao comutador na Figura 6.15? Como é possível saber que o hospedeiro se locomoveu de um AP a outro? O leitor talvez se lembre de que, no Capítulo 5, dissemos que comutadores são "autodidatas" e constroem automaticamente suas tabelas de repasse. Essa característica de autoaprendizagem funciona bem para movimentações ocasionais (por exemplo, quando um profissional é transferido de um departamento para outro); contudo, comutadores não são projetados para suportar usuários com alto grau de mobilidade, que querem manter conexões TCP enquanto se movimentam entre BSSs. Para avaliar este problema aqui, lembre-se de que, antes da movimentação, o comutador tem um registro em sua tabela de repasse que vincula o endereço MAC de H1 com a interface de saída do comutador por meio da qual o H1 pode ser alcançado. Se H1 estiver inicialmente em BSS1, então um datagrama

FIGURA 6.15 MOBILIDADE NA MESMA SUB-REDE

destinado a H1 será direcionado a ele via AP1. Contudo, tão logo H1 se associe com BSS2, seus quadros deverão ser direcionados para AP2. Uma solução (na verdade um tanto forçada) é o AP2 enviar ao comutador um quadro Ethernet de difusão com o endereço de origem de H1 logo após a nova associação. Quando o comutador recebe o quadro, atualiza sua tabela de repasse, permitindo que H1 seja alcançado via AP2. O grupo de padrões 802.11f está desenvolvendo um protocolo entre APs para cuidar dessas e outras questões associadas.

6.3.5 Recursos avançados em 802.11

Finalizaremos nossa abordagem sobre 802.11 com uma breve discussão sobre capacidades avançadas encontradas nas redes 802.11. Como veremos, essas capacidades *não* são completamente especificadas no padrão 802.11, mas, em vez disso, são habilitadas por mecanismos especificados no padrão. Isso permite que fornecedores diferentes implementem essas capacidades usando suas próprias abordagens, aparentemente trazendo vantagens sobre a concorrência.

Adaptação da taxa 802.11

Vimos antes, na Figura 6.3, que as diferentes técnicas de modulação (com as diferentes taxas de transmissão que elas fornecem) são adequadas para diferentes cenários de SNR. Considere, por exemplo, um usuário 802.11 móvel que está, inicialmente, a 20 metros de distância da estação-base, com uma alta relação sinal-ruído. Dada a alta SNR, o usuário pode se comunicar com essa estação usando uma técnica de modulação da camada física que oferece altas taxas de transmissão enquanto mantém uma BER baixa. Esse usuário é um felizardo! Suponha agora que o usuário se torne móvel, se distanciando da estação-base, e que a SNR diminui à medida que a distância da estação-base aumenta. Neste caso, se a técnica de modulação usada no protocolo 802.11 que está operando entre a estação-base e o usuário não mudar, a BER será inaceitavelmente alta à medida que a SNR diminui e, por conseguinte, nenhum quadro transmitido será recebido corretamente.

Por essa razão, algumas execuções de 802.11 possuem uma capacidade de adaptação de taxa que seleciona, de maneira adaptável, a técnica de modulação da camada física sobreposta a ser usada com base em características atuais ou recentes do canal. Se um nó enviar dois quadros seguidos sem receber confirmação (uma indicação implícita de erros de bit no canal), a taxa de transmissão cai para a próxima taxa mais baixa. Se dez quadros seguidos forem confirmados, ou se um temporizador (que registra o tempo desde o último *fallback*) expirar, a taxa de transmissão aumenta para a próxima taxa mais alta. Esse mecanismo de adaptação da taxa compartilha a mesma filosofia de "investigação" (refletida por recebimentos de ACK); quando algo "ruim" acontece, a taxa de transmissão é reduzida. A adaptação da taxa 802.11 e o controle de congestionamento TCP são, desse modo, semelhantes à criança: está sempre exigindo mais e mais de seus pais (digamos, mais doces, para uma criança, e chegar mais

tarde em casa, para adolescentes) até eles por fim dizerem "Chega!" e a criança desistir (para tentar novamente após a situação melhorar!). Diversos métodos também foram propostos para aperfeiçoar esse esquema básico de ajuste automático de taxa [Kamerman, 1997; Holland, 2001; Lacage, 2004].

Gerenciamento de energia

A energia é uma fonte preciosa em aparelhos móveis e, assim, o padrão 802.11 provê capacidades de gerenciamento de energia, permitindo que os nós 802.11 minimizem o tempo de suas funções de percepção, transmissão e recebimento, e outros circuitos necessários para "funcionar". O gerenciamento de energia 802.11 opera da seguinte maneira. Um nó é capaz de alternar claramente entre os estados "dormir" e "acordar" (como um aluno com sono em sala de aula!). Um nó indica ao ponto de acesso que entrará no modo de dormir ajustando o bit de gerenciamento de energia no cabeçalho de um quadro 802.11 para 1. Um temporizador localizado no nó é, então, ajustado para acordar o nó antes de um AP ser programado para enviar seu quadro de sinalização (lembre-se de que, em geral, um AP envia um quadro de sinalização a cada 100 ms). Uma vez que o AP descobre, pelo bit de transmissão de energia, que o nó vai dormir, ele (o AP) sabe que não deve enviar nenhum quadro àquele nó, e armazenará qualquer quadro destinado ao hospedeiro que está dormindo para transmissão posterior.

Um nó acordará logo após o AP enviar um quadro de sinalização, e logo entrará no modo ativo (ao contrário do aluno sonolento, esse despertar leva apenas 250 μs [Kamerman, 1997]!). Os quadros de sinalização enviados pelo AP contêm uma relação de nós cujos quadros foram mantidos em buffer no AP. Se não houver quadros mantidos em buffer para o nó, ele pode voltar a dormir. Caso contrário, o nó pode solicitar explicitamente o envio dos quadros armazenados, enviando uma mensagem de *polling* ao AP. Com um tempo de 100 ms entre as sinalizações, um despertar de 250 μs e um tempo semelhantemente pequeno para receber um quadro de sinalização e verificar que não haja quadros em buffer, um nó que não possui quadros para enviar ou receber pode dormir 99% do tempo, resultando em uma economia de energia significativa.

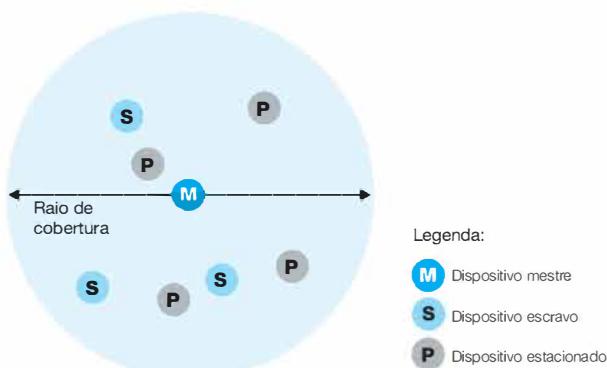
6.3.6 Redes pessoais: Bluetooth e Zigbee

Como ilustrado na Figura 6.2, o padrão Wi-Fi IEEE 802.11 é voltado para a comunicação entre aparelhos separados por até 100 m (exceto quando 802.11 é usado em configuração ponto a ponto com antena direcional). Dois outros protocolos IEEE 802 — Bluetooth e Zigbee (definidos nos padrões IEEE 802.15.1 e IEEE 802.15.4 [IEEE 802.15, 2012]) e o WiMAX (definido no padrão IEEE 802.16 [IEEE 802.16d, 2004; IEEE 802.16e, 2005]) — são padrões para se comunicar a distâncias mais curtas e mais longas, nessa ordem. Examinaremos o WiMAX rapidamente quando discutirmos sobre redes de dados celulares, na Seção 6.4; aqui, vamos nos concentrar em redes para distâncias mais curtas.

Bluetooth

Uma rede IEEE 802.15.1 opera sobre uma curta faixa, a baixa potência, e a um custo baixo. Esta é, basicamente, uma tecnologia de "substituição de cabos" de baixa velocidade, curto alcance e baixa potência para interconectar notebooks, dispositivos periféricos, telefones celulares e smartphones, enquanto 802.11 é uma tecnologia de "acesso" de velocidade mais alta, alcance médio e potência mais alta. Por essa razão, as redes 802.15.1, às vezes, são denominadas redes pessoais sem fio (WPAN — *Wireless Personal Area Networks*). As camadas de enlace e física do 802.15.1 são baseadas na especificação do **Bluetooth** anterior para redes pessoais [Held, 2001; Bisdikian, 2001]. Redes 802.15.1 operam na faixa de rádio não licenciada de 2,4 GHz em modo TDM, com intervalos de tempo (*time slots*) de 625 μs. Durante cada intervalo de tempo, um emissor transmite por um entre 79 canais, sendo que, de intervalo para intervalo, o canal muda de uma maneira conhecida, porém pseudoaleatória. Essa forma de saltar de canal em canal, conhecida como **espectro espalhado com salto de frequência** (FHSS

— *Frequency-Hopping Spread Spectrum*), espalha transmissões sobre o espectro de frequência pelo tempo. O 802.15.1 pode oferecer velocidades de dados de até 4 Mbits/s.

Redes 802.15.1 são redes *ad hoc*: não é preciso que haja uma infraestrutura de rede (por exemplo, um ponto de acesso) para interconectar dispositivos 802.15.1. Assim, esses dispositivos devem se organizar por si sós. Dispositivos 802.15.1 são primeiro organizados em uma **picorrede** (*piconet*: pequena rede) de até oito dispositivos ativos, como ilustra a Figura 6.16. Um desses dispositivos é designado como o mestre, e os outros agem como escravos. Na verdade, o nó mestre comanda a picorrede como um rei — seu relógio determina o tempo na picorrede, ele pode transmitir em cada intervalo de tempo de número ímpar e um escravo pode transmitir somente após o mestre ter se comunicado com ele no intervalo de tempo anterior e, mesmo assim, só pode transmitir para o mestre. Além dos dispositivos escravos, a rede também pode conter até 255 dispositivos estacionados. Esses dispositivos não podem se comunicar até que o nó mestre tenha mudado seus estados de estacionado para ativo.

Se o leitor quiser obter mais informações sobre WPANs 802.15.1, pode consultar as referências do Bluetooth [Held, 2001; Bisdikian, 2001] ou o site oficial do IEEE 802.15 [IEEE 802.15, 2012].

Zigbee

Uma segunda rede pessoal padronizada pelo IEEE é o padrão 802.14.5 [IEEE 802.15, 2012], conhecido como Zigbee. Enquanto as redes Bluetooth oferecem uma taxa de dados de "substituição de cabo" de mais de um Mbit por segundo, Zigbee é voltada para aplicações de menos potência, menor taxa de dados e menor ciclo de trabalho do que Bluetooth. Embora sejamos tentados a pensar que "maior e mais rápido é melhor", nem todas as aplicações de rede precisam de muita largura de banda e dos custos mais altos decorrentes disso (custos tanto econômicos quanto de energia). Por exemplo, sensores domésticos de temperatura e iluminação, dispositivos de segurança e interruptores de parede são todos dispositivos muito simples, de baixa potência, baixo ciclo de trabalho e baixo custo. Portanto, Zigbee é ideal para esses dispositivos. Zigbee define taxas de canal de 20, 40, 100 e 250 Kbits/s, dependendo da frequência do canal.

Os nós em uma rede Zigbee podem ser de dois tipos. Os chamados "dispositivos de função reduzida" operam como escravos controlados por um único "dispositivo de função completa", assim como dispositivos Bluetooth escravos. Um dispositivo de função completa pode operar como um dispositivo mestre no Bluetooth controlando vários dispositivos escravos, e múltiplos dispositivos de função completa podem, além disso, ser configurados em uma rede de malha, na qual direcionam quadros entre si. Zigbee compartilha muitos mecanismos de protocolo que já encontramos em outros protocolos da camada de enlace: quadros de sinalização e confirmações da camada de enlace (semelhantes ao 802.11), protocolos de acesso aleatório de detecção de portadora com

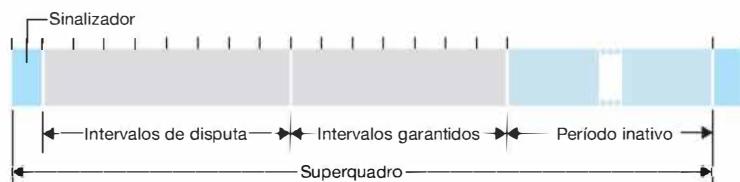



FIGURA 6.16 UMA PICORREDE BLUETOOTH

recuo exponencial binário (semelhante ao 802.11 e Ethernet) e alocação fixa, garantida, de intervalos de tempo (semelhante ao DOCSIS).

Redes Zigbee podem ser configuradas de muitas maneiras diferentes. Vamos considerar o caso simples de um único dispositivo de função completa controlando vários dispositivos de função reduzida com intervalos de tempo e usando quadros de sinalização. A Figura 6.17 mostra um caso em que a rede Zigbee divide o tempo em superquadros recorrentes, cada qual começando com um quadro de sinalização. Cada quadro de sinalização divide o superquadro em um período ativo (durante o qual os dispositivos podem transmitir) e um período inativo (durante o qual todos os dispositivos, inclusive o controlador, podem dormir e, portanto, economizar energia). O período ativo consiste em 16 intervalos de tempo, alguns usados pelos dispositivos em uma forma de acesso aleatório CSMA/CA, e alguns são alocados pelo controlador para dispositivos específicos, oferecendo assim o acesso garantido ao canal para esses dispositivos. Outros detalhes sobre redes Zigbee poderão ser encontrados em Baronti [2007], IEEE 802.15.4 [2012].

FIGURA 6.17 ESTRUTURA DO SUPERQUADRO ZIGBEE 802.14.4

6.4 ACESSO CELULAR À INTERNET

Na seção anterior vimos como um hospedeiro da Internet pode acessá-la quando estiver dentro de um *hotspot* Wi-Fi, isto é, quando estiver nas vizinhanças de um ponto de acesso 802.11. Mas a área de cobertura da maioria dos *hotspots* Wi-Fi é pequena, entre 10 e 100 m de diâmetro. O que fazer, então, quando precisamos desesperadamente de um acesso sem fio à Internet e não podemos acessar um *hotspot* Wi-Fi?

Dado que a telefonia celular agora está presente por toda parte, em muitas áreas no mundo inteiro, uma estratégia natural é estender redes celulares de modo que suportem não apenas a telefonia de voz, mas também o acesso sem fio à Internet. Idealmente, esse acesso teria uma velocidade razoavelmente alta e ofereceria mobilidade imperceptível, permitindo que usuários mantivessem suas sessões TCP enquanto viajassem, por exemplo, em um ônibus ou em um trem. Com taxas de bits altas o bastante entre a origem e o usuário e vice-versa, o usuário poderia até mesmo manter sessões de videoconferência enquanto estivesse em trânsito. Esse cenário não é assim tão implausível. Em 2012, muitas operadoras de telefonia celular ofereciam a seus assinantes um serviço de acesso celular à Internet por menos de 50 dólares mensais com taxas de bits entre a operadora e o usuário e vice-versa na faixa de algumas centenas de kilobits por segundo. Taxas de dados de vários megabits por segundo estão se tornando acessíveis à medida que os serviços de dados de banda larga, como aqueles que veremos aqui, são cada vez mais empregados.

Nesta seção, damos uma breve visão geral das tecnologias de acesso celular à Internet já existentes e emergentes. Mais uma vez, aqui focalizaremos o primeiro salto sem fio, bem como a rede que conecta o primeiro salto sem fio à rede telefônica maior e/ou à Internet; na Seção 6.7 consideraremos como as chamadas são roteadas para um usuário que está se movimentando entre estações-base. Nossa breve discussão oferecerá, necessariamente, apenas uma descrição simplificada e de alto nível das tecnologias celulares. É claro que a questão das comunicações celulares modernas é de grande amplitude e profundidade, e muitas universidades oferecem diversos cursos sobre o assunto. O leitor que desejar um conhecimento mais profundo da questão deve consultar Goodman [1997]; Kaaranen [2001]; Lin [2001]; Korhonen [2003]; Schiller [2003]; Scourias [2012]; Turner [2012]; Akyildiz [2010], bem como as referências particularmente excelentes e completas em Mouly [1992].

6.4.1 Visão geral da arquitetura de rede celular

Em nossa discussão sobre a arquitetura da rede celular nesta seção, adotaremos a terminologia dos padrões do *Sistema Global para Comunicações Móveis* (GSM — *Global System for Mobile Communications*). (Para os amantes de história, a sigla GSM, na origem, veio de *Groupe Spécial Mobile*, até a forma anglicizada ser adotada, mantendo as letras iniciais originais.) Nos anos 1980, os europeus reconheceram a necessidade de um sistema pan-europeu de telefonia celular digital que substituiria os diversos sistemas analógicos de telefonia celular incompatíveis, levando ao padrão GSM [Mouly, 1992]. Os europeus implantaram a tecnologia GSM com um grande sucesso no início dos anos 1990 e, desde então, o GSM cresceu e se tornou um gigante do mundo da telefonia celular, com mais de 80% de assinantes no mundo utilizando essa tecnologia.

Quando as pessoas falam sobre tecnologia celular, em geral a classificam como pertencendo a uma das diversas "gerações". As primeiras gerações foram, em essência, projetadas para o tráfego de voz. Os sistemas de primeira geração (1G) eram sistemas FDMA analógicos, desenvolvidos especialmente para a comunicação apenas por voz. Esses sistemas 1G quase não existem mais, tendo sido substituídos pelos sistemas digitais 2G. Os sistemas originais 2G também foram projetados para voz, sendo estendidos mais tarde (2,5G) para suportar dados (por exemplo, a Internet), bem como o serviço de voz. Os sistemas 3G, que estão sendo executados hoje, também suportam voz e dados, mas com uma ênfase cada vez maior nas capacidades de dados e enlaces de acesso via rádio com maior velocidade.

HISTÓRIA

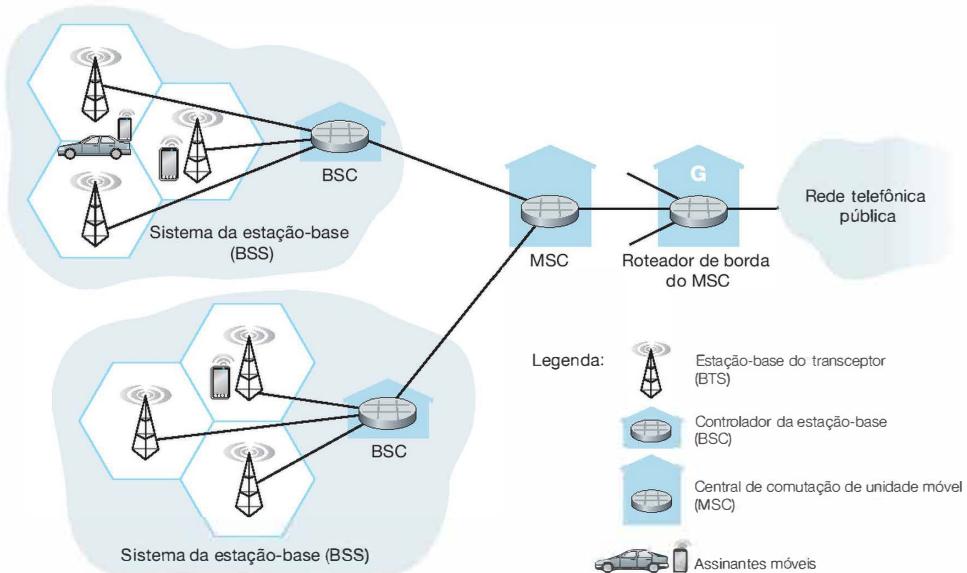
Celular móvel 3G *versus* LANs sem fio

Muitas operadoras de telefonia celular móvel estão disponibilizando sistemas celulares móveis 3G, com taxas de dados de 2 Mbits/s em ambiente fechado e 384 kbits/s, ou mais, ao ar livre. Esses sistemas 3G estão sendo disponibilizados em faixas de radiofrequência licenciadas, e o preço que algumas operadoras pagam aos governos pela licença é uma quantia considerável. Os sistemas 3G permitirão que os usuários acessem a Internet a partir de locais externos remotos enquanto estão em trânsito, de modo semelhante ao acesso por telefone celular existente hoje. Por exemplo, a tecnologia 3G permite que um usuário acesse informações sobre mapas de estradas enquanto dirige seu carro, ou informações sobre a programação de cinema enquanto estiver tomando sol na praia. Não obstante, pode-se questionar a extensão à qual os sistemas 3G serão usados, dados o seu custo e o fato de que os usuários em geral possuem acesso simultâneo a LANs sem fio e 3G:

- A infraestrutura emergente de LAN sem fio logo estará presente em quase toda parte. As LANs sem fio IEEE 802.11, que operam a taxas de 54 Mbits/s, estão sendo amplamente disponibilizadas. Quase todos os computadores portáteis e smartphones já vêm da fábrica equipados com capacidades de LAN 802.11. Além disso, as novidades em equipamentos de Internet — como câmeras sem fio e molduras para fotografias — também terão capacidades de LAN sem fio de baixa potência.
- Estações-base de LANs sem fio também podem manipular equipamentos de telefonia móvel. Muitos telefones já podem se conectar com a rede de telefonia celular ou a uma rede IP de forma nativa ou usando o serviço de voz sobre IP semelhante ao Skype, ultrapassando, assim, os serviços de dados 3G e de voz por celular da operadora.

É claro que muitos outros especialistas acham que a tecnologia 3G não só será o maior sucesso, mas também revolucionará drasticamente o modo como trabalhamos e vivemos. É claro que ambas, Wi-Fi e 3G, podem se tornar tecnologias sem fio dominantes, com equipamentos sem fio em trânsito que selecionam automaticamente a tecnologia de acesso que ofereça o melhor serviço em seu local físico atual.

Arquitetura de rede celular, 2G: conexões por voz à rede telefônica

O termo *celular* refere-se ao fato de que uma área geográfica é dividida em várias áreas de cobertura geográfica, conhecidas como células, ilustradas como hexágonos à esquerda da Figura 6.18. Assim como estudamos o padrão Wi-Fi 802.11 na Seção 6.3.1, o GSM possui sua nomenclatura específica. Cada célula contém uma estação-base de transceptor (BTS) que transmite e recebe sinais de estações móveis dentro de sua célula. A área de cobertura de uma célula depende de muitos fatores, incluindo potência da BTS, potência de transmissão de aparelhos do usuário, obstáculos na célula, como prédios, e altura das antenas da estação-base. Embora a Figura 6.18 mostre cada célula com uma estação-base de transceptor posicionada no seu meio, hoje, muitos sistemas posicionam a BTS em pontos de interseção de três células, de modo que uma única BTS com antenas direcionais possa atender a três células.

O padrão GSM para sistemas celulares 2G utiliza FDM/TDM (rádio) combinados para a interface ar. Lembre-se de que vimos no Capítulo 1 que, com FDM puro, o canal é dividido em uma série de bandas de frequência, e cada banda se dedica a uma chamada. Também no Capítulo 1, vimos que, com FDM puro, o tempo é dividido em quadros, cada quadro é dividido em intervalos e cada chamada é destinada ao uso de um intervalo específico no quadro rotativo. Nos sistemas combinados FDM/TDM, o canal é dividido em uma série de sub-bandas de frequência; dentro de cada sub-banda, o tempo é dividido em quadros e intervalos. Desse modo, para um sistema combinado FDM/TDM, se o canal for dividido em F sub-bandas e o tempo em T intervalos, então o canal poderá suportar $F \cdot T$ chamadas simultâneas. Lembre-se de que vimos, na Seção 5.3.4, que as redes de acesso a cabo também usam uma técnica combinada FDM/TDM. Os sistemas GSM consistem em bandas de frequência de 200-kHz e cada banda suporta oito chamadas TDM. O GSM codifica a voz a 13 kbits/s e 12,2 kbits/s.

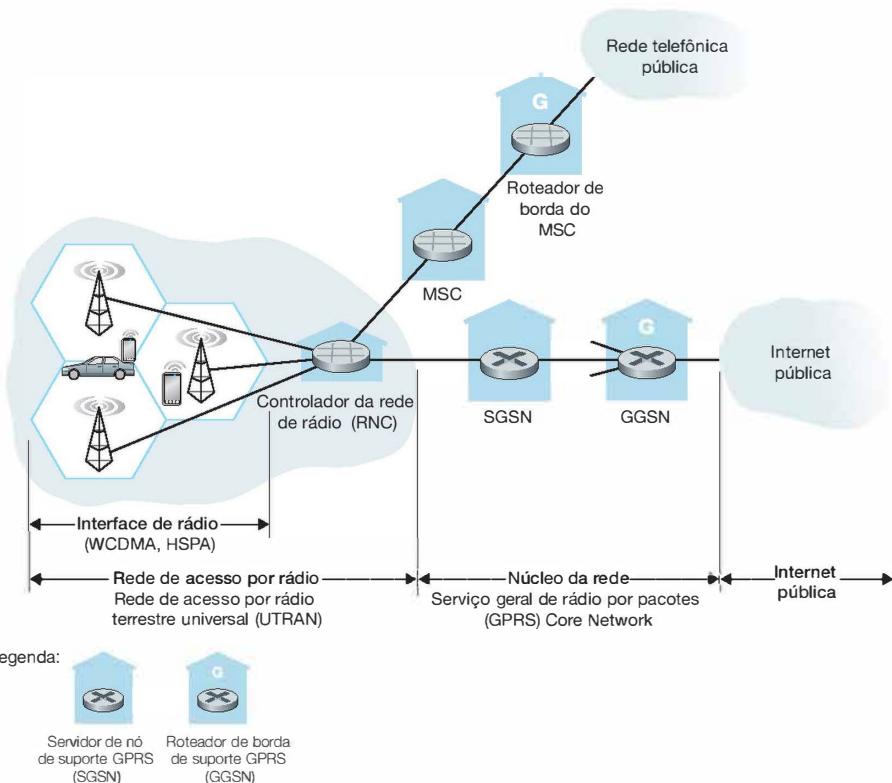
Um **controlador de estação-base (BSC)** da rede GSM normalmente prestará serviço a dezenas de estações -base do transceptor. O papel da BSC é alocar os canais de rádio da BTS a assinantes móveis, realizar **paginação** (encontrar a célula na qual reside um usuário móvel) e realizar transferência de usuários móveis — um assunto que abordaremos, em poucas palavras, na Seção 6.7.2. O controlador de estação-base e suas estações-base de transceptor, coletivamente, constituem um **sistema de estação-base (BSS)** GSM.

Como veremos na Seção 6.7, a **central de comutação móvel (MSC)** desempenha o papel central na contabilidade e autorização do usuário (por exemplo, determinar se um aparelho móvel tem permissão para se conectar à rede celular), estabelecimento e interrupção de chamada, e transferências. Em geral, uma única MSC conterá até cinco BSCs, resultando em cerca de 200K assinantes por MSC. A rede de um provedor de celular terá diversas MSCs, com MSCs especiais conhecidas como roteadores de borda das MSCs, conectando a rede celular do provedor à rede telefônica pública mais ampla.

6.4.2 Redes de dados celulares 3G: estendendo a Internet para assinantes de celular

Nossa discussão na Seção 6.4.1 foi concentrada na conexão de usuários de voz celular à rede telefônica pública. Porém, claro, quando estamos nos movimentando, também queremos ler e-mails, acessar a Web, obter serviços dependentes do local (por exemplo, mapas e recomendações de restaurantes) e talvez ainda assistir a vídeos. Para isso, nosso smartphone precisará executar uma pilha de protocolos TCP/IP completa (incluindo as camadas física, de enlace, rede, transporte e aplicação) e conectar-se à Internet por meio da rede de dados celular. O assunto de redes de dados celulares é uma coleção um tanto confusa de padrões concorrentes e em evolução, à medida que uma geração (e meia geração) substitui a anterior e introduz novas tecnologias e serviços, com novos acrônimos. Para piorar as coisas ainda mais, não há um único órgão oficial que defina os requisitos para as tecnologias 2,5G, 3G, 3,5G ou 4G, tornando mais difícil apontar as diferenças entre os padrões concorrentes. Em nossa discussão a seguir, vamos nos concentrar nos padrões 3G do UMTS (*Universal Mobile Telecommunications*

FIGURA 6.18 COMPONENTES DA ARQUITETURA DE REDE CELULAR 2G GSM

Service), desenvolvidos pelo Projeto de Parceria da 3ª Geração (3GPP — *3rd Generation Partnership Project*) [3GPP 2012], uma tecnologia 3G bastante difundida.

Vamos fazer uma análise de cima para baixo da arquitetura de rede de dados celular 3G mostrada na Figura 6.19.

O núcleo da rede 3G

O núcleo da rede de dados celular 3G conecta as redes de acesso por rádio à Internet pública. O núcleo da rede opera em conjunto com os componentes da rede celular de voz existente (particularmente, o MSC), que encontramos anteriormente na Figura 6.18. Dada a quantidade considerável de infraestrutura existente (e serviços lucrativos!) na rede celular de voz, o método utilizado pelos projetistas dos serviços de dados 3G é evidente: *deixe o núcleo da rede celular de voz GSM intacto, acrescentando funcionalidade de dados por celular em paralelo à rede de voz existente.* A alternativa — integrar novos serviços de dados diretamente no núcleo da rede celular de voz — teria gerado os mesmos desafios encontrados na Seção 4.4.4, na qual discutimos sobre a integração das tecnologias IPv6 (nova) e IPv4 (legada) na Internet.

Existem dois tipos de nós no núcleo da rede 3G: **Servidor de Nó de Suporte GPRS (SGSN –** *Serving GPRS Support Nodes*) e **Roteador de borda de suporte GPRS (GGSN —** *Gateway GPRS Support Nodes*). (O acrônimo GPRS significa *Generalized Packet Radio Service* — serviço de rádio por pacotes generalizado —, um antigo serviço celular de dados em redes 2G; aqui, discutimos a versão aperfeiçoada do GPRS nas redes 3G.) Um SGSN é responsável por entregar datagramas de e para os nós móveis na rede de acesso por rádio à qual o SGSN está ligado. O SGSN interage com o MSC da rede celular de voz para essa área, oferecendo autorização do usuário e transferência, mantendo informações de local (célula) sobre nós móveis ativos e realizando repasse de datagramas entre os nós móveis na rede de acesso por rádio e um GGSN. O GGSN atua como um roteador de borda (*gateway*), conectando vários SGSNs à Internet maior. Um GGSN, portanto, é a última parte da infraestrutura 3G que um datagrama originado do nó móvel encontra antes de entrar na Internet. Para o

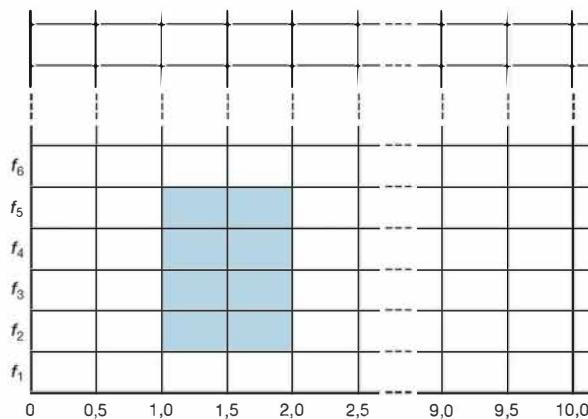
FIGURA 6.19 ARQUITETURA DO SISTEMA 3G

mundo exterior, o GGSN é semelhante a qualquer outro roteador de borda; a mobilidade dos nós 3G dentro da rede do GGSN fica oculta do mundo exterior, por trás do GGSN.

A rede de acesso por rádio 3G: a borda sem fio

A **rede de acesso por rádio** 3G é a rede do primeiro salto sem fio que vemos como usuários do 3G. O **controlador da rede de rádio (RNC)** em geral controla várias estações-base transceptoras da célula, semelhantes às estações-base que encontramos nos sistemas 2G (mas, no linguajar do UMTS 3G, conhecida oficialmente como "Node Bs" — um nome nada descritivo!). O enlace sem fio de cada célula opera entre os nós móveis e uma estação-base transceptora, assim como nas redes 2G. O RNC se conecta à rede de voz do celular por comutação de circuitos via um MSC, e à Internet por comutação de pacotes via um SGSN. Assim, embora os serviços de voz e dados por celular 3G utilizem núcleos de rede diferentes, eles compartilham uma rede comum de acesso por rádio do primeiro e último salto.

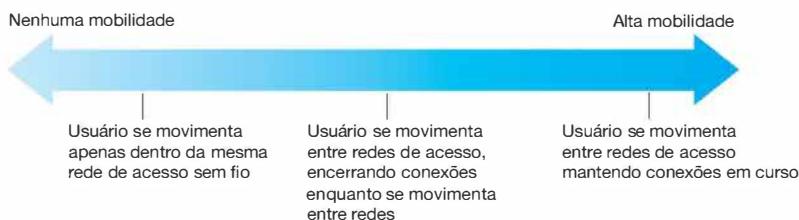
Uma mudança significativa no UMTS 3G em relação às redes 2G é que, em vez de usar o método FDMA/TDMA do GSM, o UMTS emprega uma técnica CDMA denominada *Direct Sequence Wideband* CDMA (CDMA banda larga de sequência direta) (DS-WCDMA) [Dahlman, 1998] dentro de intervalos TDMA; estes, por sua vez, são acessíveis em frequências múltiplas — uma utilização interessante de todos os três métodos de compartilhamento de canal dedicado que identificamos no Capítulo 5 e semelhante à técnica usada nas redes de acesso a cabo (veja Seção 5.3.4). Essa mudança exige uma nova rede celular 3G de acesso sem fio operando paralelamente

com a rede de rádio BSS 2G mostrada na Figura 6.18. O serviço de dados associado à especificação do WCDMA é conhecido como Acesso a Pacote em Alta Velocidade (HSP — *High Speed Packet Access*) e promete taxas de dados de até 14 Mbits/s. Mais detalhes referentes às redes 3G podem ser encontrados no site do Projeto de Parceria da 3ª Geração (3GPP) [3GPP, 2012].

6.4.3 No caminho para o 4G: LTE

Com os sistemas 3G agora sendo utilizados no mundo inteiro, será que os sistemas 4G estão bem atrasados? Decerto não! Na realidade, projeto, teste e implantação inicial dos sistemas 4G já estão sendo realizados. O padrão 4G Long-Term Evolution (LTE) apresentado pelo 3GPP tem duas inovações importantes em relação aos sistemas 3G:

- **Núcleo de pacote desenvolvido (EPC —** *Evolved Packet Core*) [3GPP Network Architecture, 2012]. O EPC é uma rede de núcleo simplificado em IP, que unifica a rede celular de voz comutada por circuitos separada e a rede celular de dados comutada por pacotes mostrada na Figura 6.19. Esta é uma rede "toda em IP", pois voz e dados serão transportados em datagramas IP. Como vimos no Capítulo 4 e estudaremos com mais detalhes no Capítulo 7, o modelo de serviço de "melhor esforço" do IP não é inerentemente bem adequado para os requisitos de desempenho exigentes do tráfego VoIP (voz sobre IP), a menos que os recursos da rede sejam controlados com cuidado para evitar (em vez de reagira a) congestionamento. Assim, uma tarefa importante do EPC é controlar os recursos da rede para oferecer essa alta qualidade de serviço. O EPC também faz uma separação nítida entre os planos de controle da rede e dados do usuário, com muitos dos recursos de suporte a mobilidade que estudaremos na Seção 6.7 sendo executados no plano de controle. O EPC permite que vários tipos de redes de acesso por rádio, incluindo redes de acesso 2G e 3G legadas, se conectem ao núcleo da rede. Duas introduções bastante claras ao EPC são [Motorola, 2007; Alcatel-Lucent, 2009].
- **Rede de acesso por rádio LTE.** O padrão LTE usa uma combinação de multiplexação por divisão de frequência e multiplexação por divisão de tempo no canal descendente, conhecida como multiplexação por divisão de frequência ortogonal (OFDM) [Rohde, 2008; Ericsson, 2011]. (O termo "ortogonal" vem do fato de que os sinais enviados em diferentes canais de frequência são criados de modo que interfiram muito pouco uns nos outros, mesmo quando as frequências de canal são pouco espaçadas.) No LTE, cada nó móvel ativo recebe um ou mais intervalos de tempo de 0,5 ms em uma ou mais das frequências do canal. A Figura 6.20 mostra a alocação de oito intervalos de tempo sobre quatro frequências. Recebendo cada vez mais intervalos de tempo (seja na mesma frequência ou em frequências diferentes), um nó móvel pode alcançar velocidades de transmissão cada vez mais altas. A (re)alocação de intervalo entre os nós móveis pode ser realizada até mesmo a cada milissegundo. Diferentes esquemas de modulação também podem ser usados para alterar a taxa de transmissão; veja nossa discussão anterior da Figura 6.3 e a seleção dinâmica de esquemas de modulação em redes Wi-Fi. Outra inovação na rede de rádio LTE é o uso de sofisticadas antenas de entrada múltipla, saída múltipla (MIMO). A taxa de dados máxima para um usuário LTE é 100 Mbits/s na direção descendente e 50 Mbits/s na direção ascendente, quando estiver usando 20 MHz do espectro sem fio.

A alocação em particular de intervalos de tempo a nós móveis não é exigida pelo padrão LTE. Em vez disso, a decisão de quais nós móveis terão permissão para transmitir em dado intervalo de tempo em dada frequência é determinada pelos algoritmos de escalonamento fornecidos pelo fornecedor de equipamento LTE e/ou operador da rede. Com o escalonamento oportunista [Bender, 2000; Kolding, 2003; Kulkarni, 2005], combinando o protocolo da camada física e as condições do canal entre remetente e destinatário e escolhendo os destinatários aos quais os pacotes serão enviados com base nas condições do canal, permite que o controlador da rede de rádio faça o melhor uso do meio sem fio. Além disso, as prioridades do usuário e os níveis de

FIGURA 6.20 VINTE INTERVALOS DE 0,5 MILISSEGUNDOS ORGANIZADOS EM QUADROS DE 10 MILISSEGUNDOS EM CADA FREQUÊNCIA. A REGIÃO SOMBREADA INDICA UMA ALOCAÇÃO DE OITO INTERVALOS

serviço contratados (por exemplo, prata, ouro ou platina) podem ser usados no escalonamento das transmissões descendentes de pacotes. Além das capacidades do LTE descritas aqui, LTE-Advanced permite larguras de banda descendentes de centenas de Mbits/s, com a alocação de canais agregados a um nó móvel [Akyildiz, 2010].

Outra tecnologia 4G sem fio — WiMAX (World Interoperability for Microwave Access) — é uma família de padrões IEEE 802.16 que diferem bastante do LTE. Ainda não sabemos se a tecnologia 4G preferida será LTE ou WiMAX, mas no momento (2012), LTE parece ter um impulso mais significativo. Uma discussão detalhada do WiMAX poderá ser encontrada no site deste livro.

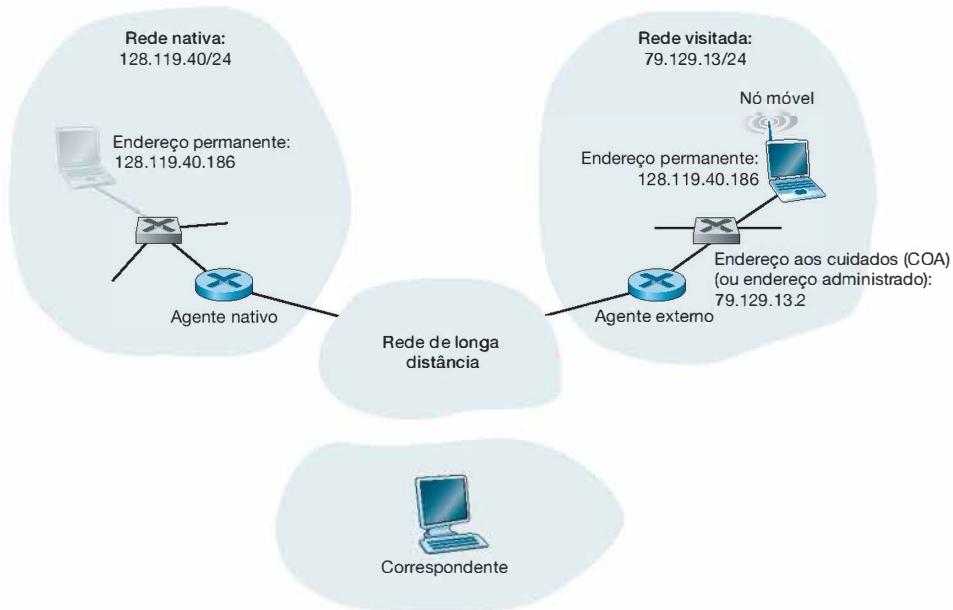
6.5 GERENCIAMENTO DA MOBILIDADE: PRINCÍPIOS

Após estudarmos a natureza *sem fio* dos enlaces de comunicação em uma rede sem fio, é hora de voltarmos nossa atenção à *mobilidade* que esses enlaces sem fio possibilitam. No sentido mais amplo, um nó móvel é aquele que muda seu ponto de conexão com a rede ao longo do tempo. Como o termo *mobilidade* adquiriu muitos significados nos campos da computação e da telefonia, será proveitoso, antes de tudo, considerarmos diversas dimensões da mobilidade com algum detalhe.

- *Do ponto de vista da camada de rede, até que ponto um usuário é móvel?* Um usuário fisicamente móvel apresentará à camada de rede conjuntos de desafios muito diferentes dependendo de como ele se movimenta entre pontos de conexão com a rede. Em uma extremidade do espectro na Figura 6.21, um usuário pode carregar consigo um notebook equipado com uma placa de interface de rede sem fio dentro de um prédio. Como vimos na Seção 6.3.4, esse usuário *não* é móvel do ponto de vista da camada de rede. Além do mais, se ele se associar com o mesmo ponto de acesso independentemente do local, então não será móvel nem mesmo do ponto de vista da camada de enlace.
- Na outra extremidade do espectro, considere o usuário que está dentro de um BMW, correndo pela estrada a 150 km/h, passando por várias redes de acesso sem fio e querendo manter uma conexão TCP ininterrupta com uma aplicação remota durante a viagem. Esse usuário é *definitivamente* móvel! Entre esses extremos está um usuário que leva seu notebook de um local (por exemplo, escritório ou quarto de dormir) a outro (por exemplo, lanchonete, biblioteca) e quer se conectar à rede no novo local. Ele também é móvel (embora menos do que o motorista da BMW!), mas não precisa manter uma conexão

FIGURA 6.21 VÁRIOS GRAUS DE MOBILIDADE DO PONTO DE VISTA DA CAMADA DE REDE

ativa enquanto se movimenta entre pontos de conexão com a rede. A Figura 6.21 ilustra esse espectro de mobilidade do usuário do ponto de vista da camada de rede.

- *Qual é a importância de o endereço do nó móvel permanecer sempre o mesmo?* Com a telefonia móvel, o número de seu telefone — basicamente, o endereço de camada de rede do seu aparelho — permanece o mesmo quando você transita entre uma operadora de rede de telefonia móvel e outra. Um notebook também terá de manter o mesmo endereço IP enquanto se movimenta entre redes IP?
  A resposta a essa pergunta dependerá muito da aplicação que está sendo executada. Para o motorista da BMW que quer manter uma conexão TCP ininterrupta com uma aplicação remota enquanto voa pela estrada, seria conveniente manter o mesmo endereço IP. Lembre-se de que dissemos, no Capítulo 3, que uma aplicação de Internet precisa conhecer o endereço IP e o número de porta da entidade remota com a qual está se comunicando. Se uma entidade móvel puder manter seu endereço IP enquanto estiver em trânsito, a mobilidade torna-se invisível do ponto de vista da aplicação. Essa transparência é de grande valor — uma aplicação não precisa se preocupar com uma potencial mudança de endereço IP e o mesmo código de aplicação servirá igualmente a conexões móveis e não móveis. Na próxima seção veremos que o IP móvel oferece essa transparência, permitindo que um nó móvel mantenha seu endereço IP permanente enquanto se movimenta entre redes.
  Por outro lado, um usuário móvel menos sofisticado poderia querer apenas desligar seu notebook no escritório, levá-lo para casa, ligá-lo de novo e continuar trabalhando. Se o notebook funciona principalmente como um cliente em aplicações cliente-servidor (por exemplo, enviar/receber e-mails, navegar pela web, usar Telnet com um hospedeiro remoto), o endereço IP particular utilizado pelo notebook não é tão importante. Em particular, é possível passar muito bem com um endereço temporariamente alocado ao notebook pelo ISP que atende a residência. Na Seção 4.4, vimos que o DHCP já oferece essa funcionalidade.
- *Qual é a infraestrutura cabeada de suporte disponível?* Em todos os quatro cenários descritos, admitimos de modo implícito haver uma infraestrutura fixa à qual o usuário móvel pode se conectar — por exemplo, a rede do ISP que atende a residência, a rede de acesso sem fio no local de trabalho ou as redes de acesso sem fio na rodovia. E se tal infraestrutura não existir? Se dois usuários estiverem a uma distância que permita comunicação, eles poderão estabelecer uma conexão de rede na ausência de qualquer outra infraestrutura de camada de rede? As redes *ad hoc* oferecem exatamente essas capacidades. Essa área, que está em rápido desenvolvimento e representa a pesquisa de ponta em redes móveis, está fora do escopo deste livro. Perkins [2000] e as páginas Web do grupo de trabalho Mobile *Ad hoc* Network (manet) do IETF apresentam um tratamento aprofundado do assunto.

Para ilustrar as questões que envolvem permitir que um usuário mantenha conexões em curso enquanto se movimenta entre redes, vamos fazer uma analogia com seres humanos. Um adulto de 20 e poucos anos que sai da casa dos pais torna-se móvel, pois passa a morar em uma série de quartos e/ou apartamentos e está sempre mudando de endereço. Se uma velha amiga quiser entrar em contato com ele, como conseguirá o endereço de seu amigo móvel? Uma maneira comum de fazer isso é entrar em contato com a família, já que um jovem

móvel costuma informar seus novos endereços (nem que seja só para que os pais possam lhe enviar dinheiro para ajudar a pagar o aluguel!). A residência da família, com seu endereço permanente, torna-se o único lugar a que outros podem se dirigir como uma primeira etapa para estabelecer comunicação com o jovem móvel. As comunicações posteriores da velha amiga podem ser indiretas (por exemplo, pelo envio de uma carta primeiro à casa dos pais, que a encaminharão ao jovem móvel) ou diretas (por exemplo, utilizar o endereço informado pelos pais e enviar uma carta diretamente ao amigo móvel).

Em um ambiente de rede, a residência permanente de um nó móvel (tal como um notebook ou um smartphone) é conhecida como **rede nativa** (*home network*) e a entidade dentro dessa rede que executa o gerenciamento de funções de mobilidade em nome do nó móvel é conhecida como **agente nativo** (*home agent*). A rede na qual o nó móvel está residindo é conhecida como **rede externa** (*foreign network*) ou **rede visitada** (*visited network*), e a entidade dentro da rede externa que auxilia o nó móvel no gerenciamento das funções de mobilidade discutidas adiante é conhecida como **agente externo** (*foreign agent*). No caso de profissionais móveis, suas redes nativas possivelmente seriam as redes das empresas em que trabalham, enquanto a rede visitada poderia ser a rede de um colega que estão visitando. Um **correspondente** é a entidade que quer se comunicar com o nó móvel. A Figura 6.22 ilustra esses conceitos, bem como conceitos de endereçamento considerados mais adiante. Observe que, na Figura 6.22, os agentes são colocados junto dos roteadores (por exemplo, como processos que rodam em roteadores), mas, como alternativa, poderiam estar rodando em outros hospedeiros ou servidores na rede.

FIGURA 6.22 ELEMENTOS INICIAIS DE UMA ARQUITETURA DE REDE MÓVEL

6.5.1 Endereçamento

Já observamos que, para a mobilidade do usuário ser transparente para aplicações de rede, é desejável que um nó móvel mantenha seu endereço quando transita de uma rede para outra. Quando um nó móvel residir em uma rede externa, todo o tráfego enviado ao endereço permanente do nó agora precisará ser roteado para a rede

externa. Como isso pode ser feito? Uma opção é a rede externa anunciar para todas as outras que o nó móvel agora reside em sua rede, o que poderia ser feito mediante a costumeira troca de informações de roteamento interdomínio e intradomínio, exigindo poucas mudanças na infraestrutura de roteamento. A rede externa poderia apenas anunciar a seus vizinhos que tem uma rota altamente específica para o endereço permanente do nó móvel (isto é, em essência, informar a outras redes que ela tem o caminho correto para rotear datagramas para o endereço permanente do nó móvel; ver Seção 4.4). Esses vizinhos então propagariam a informação de roteamento por toda a rede como parte do procedimento normal de atualização de informações de roteamento e tabelas de repasse. Quando o nó móvel saísse de uma rede externa e se juntasse a outra, a nova rede externa anunciaria uma nova rota, altamente específica, até o nó móvel e a antiga rede externa retiraria suas informações de roteamento referentes ao nó móvel.

Esse procedimento resolve dois problemas de uma vez só e o faz sem promover mudanças significativas na infraestrutura da camada de rede. Outras redes conhecem a localização do nó móvel e é fácil rotear datagramas para o nó móvel, visto que as tabelas de repasse dirigirão datagramas à rede externa. Uma desvantagem significativa, contudo, é a da facilidade de expansão. Se a responsabilidade pelo gerenciamento da mobilidade tivesse de recair sobre os roteadores da rede, eles teriam de manter registros em tabelas de repasse para potencialmente milhões de nós móveis e atualizar esses registros à medida que os nós se movimentassem. Algumas desvantagens adicionais serão exploradas nos problemas ao final deste capítulo.

Um método alternativo (que tem sido adotado na prática) é passar a funcionalidade de mobilidade do núcleo da rede para a borda — um tema recorrente em nosso estudo da arquitetura da Internet. Um modo natural de fazer isso é por meio da rede nativa do nó móvel. De maneira muito semelhante ao modo como os pais daquele jovem de 20 e poucos anos monitoram a localização do filho, o agente nativo na rede nativa do nó móvel pode monitorar a rede externa na qual o nó móvel reside. Decerto será preciso um protocolo (ou um agente externo representando o nó móvel) entre o nó móvel e o agente nativo para atualizar a localização do nó móvel.

Agora vamos considerar o agente externo com mais detalhes. A técnica conceitualmente mais simples, mostrada na Figura 6.22, é localizar agentes externos nos roteadores de borda na rede externa. Um dos papéis do agente externo é criar o denominado **endereço aos cuidados** (*care-of-address* — **COA**) **ou endereço administrado** para o nó móvel, sendo que a parte da rede do endereço COA combinaria com a parte da rede do endereço da rede externa. Assim, há dois endereços associados a um nó móvel, seu **endereço permanente** (semelhante ao endereço da família do nosso jovem móvel) e seu endereço COA, às vezes denominado **endereço externo** (semelhante ao endereço da casa onde nosso jovem móvel estiver residindo). No exemplo da Figura 6.22, o endereço permanente do nó móvel é 128.119.40.186. Quando está visitando a rede 79.129.13/24, o nó móvel tem um COA 79.129.13.2. Um segundo papel desempenhado pelo agente externo é informar ao agente nativo que o nó móvel está residindo em sua rede (a rede do agente externo) e tem o endereço COA informado. Veremos, em breve, que o COA pode ser utilizado para redirecionar datagramas para o nó móvel através de seu agente externo.

Embora tenhamos separado a funcionalidade do nó móvel e do agente externo, vale a pena observar que o nó móvel também pode assumir as responsabilidades do agente externo. Por exemplo, o nó móvel poderia obter um COA na rede externa (por exemplo, utilizando um protocolo como o DHCP) e ele mesmo informar seu COA ao agente nativo.

6.5.2 Roteamento para um nó móvel

Agora já vimos como um nó móvel obtém um COA e como é possível informar esse endereço ao agente nativo. Mas conseguir que o agente nativo conheça o COA resolve apenas parte do problema. Como datagramas devem ser endereçados e repassados para o nó móvel? Visto que só o agente nativo (e não os roteadores no âmbito da rede) conhece a localização do nó móvel, já não será mais suficiente apenas endereçar um datagrama para o endereço permanente do nó móvel e enviá-lo para a infraestrutura da camada de rede. É preciso fazer algo mais. Duas técnicas podem ser identificadas, as quais denominaremos roteamento indireto e roteamento direto.

Roteamento indireto para um nó móvel

Vamos considerar primeiro um correspondente que quer enviar um datagrama a um nó móvel. Na abordagem de **roteamento indireto** o correspondente apenas endereça o datagrama ao endereço permanente do nó móvel, envia o datagrama para a rede e nem precisa saber se o nó móvel reside em sua rede nativa ou está visitando uma rede externa; assim, a mobilidade é completamente transparente para o correspondente. Esses datagramas são primeiro roteados, como sempre, para a rede local do nó móvel. Isso é ilustrado na etapa 1 da Figura 6.23.

Agora vamos voltar nossa atenção ao agente nativo. Além de ser responsável por interagir com um agente externo para monitorar o COA do nó móvel, ele tem outra função muito importante. Sua segunda tarefa é ficar à espreita de datagramas que chegam e são endereçados a nós cuja rede nativa é a rede do agente nativo, mas que estão residindo no momento em uma rede externa. O agente nativo intercepta esses datagramas e então os repassa a um nó móvel por um processo de duas etapas. Primeiro, o datagrama é repassado para o agente externo usando o COA do nó móvel (etapa 2 na Figura 6.23), e depois do agente externo para o nó móvel (etapa 3 na Figura 6.23).

É instrutivo considerar esse redirecionamento com mais detalhes. O agente nativo precisará endereçar o datagrama usando o COA do nó móvel, de modo que a camada de rede roteará o datagrama para a rede externa. Por outro lado, é desejável deixar intacto o datagrama do correspondente, pois a aplicação que recebe o datagrama deve desconhecer que este foi repassado por meio do agente nativo. Ambas as metas podem ser cumpridas fazendo o agente nativo **encapsular** o datagrama original completo do correspondente dentro de um novo (e maior) datagrama. Este é endereçado e entregue ao COA do nó móvel. O agente externo "proprietário" do COA receberá e desencapsulará o datagrama — isto é, removerá o datagrama original do correspondente de dentro daquele datagrama maior de encapsulamento e repassará o datagrama original (etapa 3 na Figura 6.23) para o nó móvel. A Figura 6.24 mostra um datagrama original de um correspondente sendo enviado para a rede nativa, um datagrama encapsulado sendo enviado ao agente externo e o datagrama original sendo entregue ao nó móvel. O leitor atento notará que o encapsulamento/desencapsulamento descrito aqui é idêntico à noção de implementação de túnel discutida no Capítulo 4, no contexto do IP de transmissão para um grupo (*multicast*) e do IPv6.

FIGURA 6.23 ROTEAMENTO INDIRETO PARA UM NÓ MÓVEL

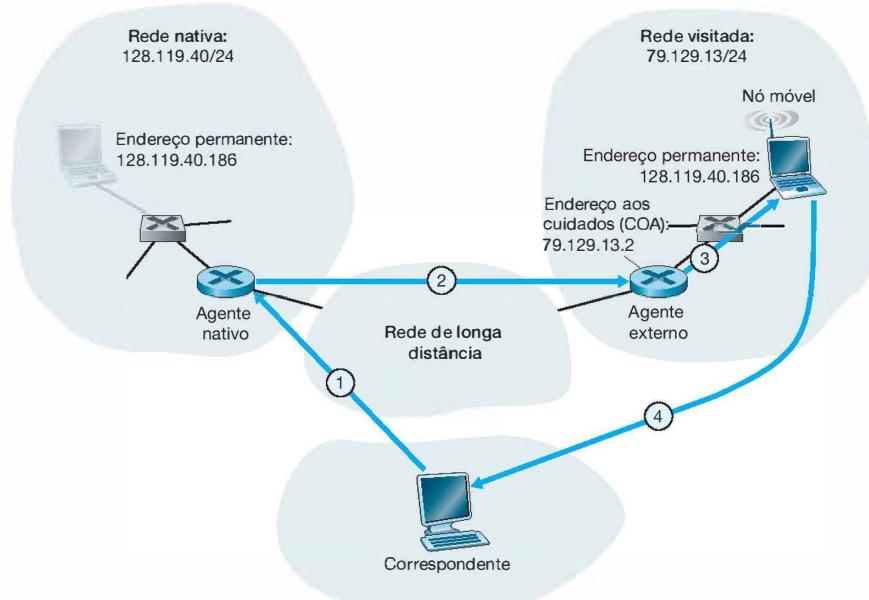
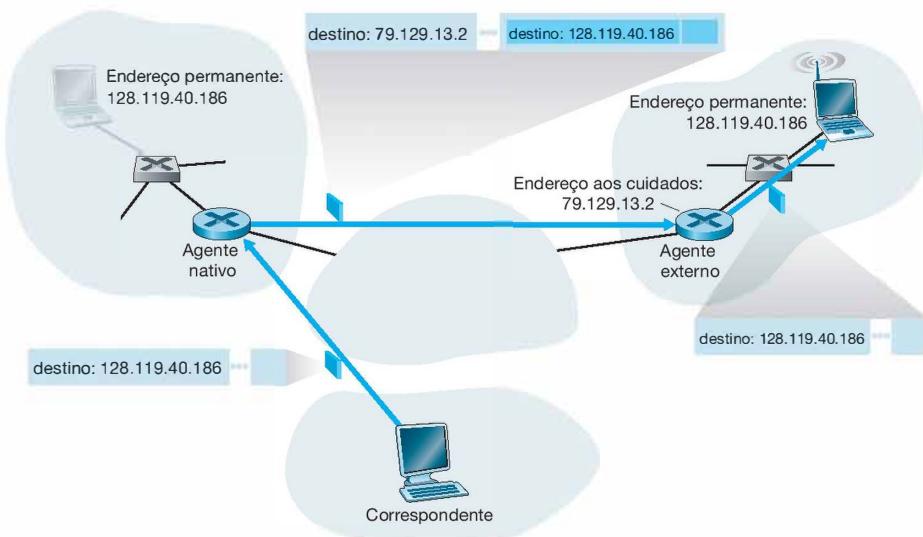



FIGURA 6.24 ENCAPSULAMENTO E DESENCAPSULAMENTO

A seguir, vamos considerar como um nó móvel envia datagramas a um correspondente. Isso é muito simples, pois o nó móvel pode endereçar seu datagrama *diretamente* ao correspondente (usando o próprio endereço permanente como o de origem e o do correspondente como o endereço de destino). Visto que o nó móvel conhece o endereço do correspondente, não há necessidade de rotear o datagrama de volta por meio do agente nativo. Isso é mostrado como etapa 4 na Figura 6.23.

Vamos resumir o que discutimos sobre roteamento indireto relacionando as novas funcionalidades da camada de rede exigidas para dar suporte à mobilidade.

- *Um protocolo de nó móvel ao agente externo.* O nó móvel fará seu registro no agente externo ao se conectar à rede externa. De modo semelhante, um nó móvel cancelará seu registro no agente externo quando sair da rede externa.
- *Um protocolo de registro do agente externo ao agente nativo.* O agente externo registrará o COA do nó móvel no agente nativo. Um agente externo não precisa cancelar explicitamente um COA quando um nó móvel sai da rede porque o registro subsequente de um novo COA se encarregará disso quando o nó móvel passar para uma nova rede.
- *Um protocolo de encapsulamento de datagrama para o agente nativo.* Encapsulamento e repasse do datagrama original do correspondente dentro de um datagrama endereçado ao COA.
- *Um protocolo de desencapsulamento para o agente externo.* Extração do datagrama original do correspondente de dentro daquele que o encapsulou e repasse do datagrama original ao nó móvel.

A discussão anterior provê todas as peças — agentes externos, agente nativo e repasse indireto — de que um nó móvel necessita para manter uma conexão em curso enquanto transita entre redes. Para ilustrar como essas peças se encaixam, suponha que o nó móvel está ligado à rede externa A, registrou um COA no agente local na rede A e está recebendo datagramas que estão sendo roteados indiretamente por meio de seu agente nativo. O nó móvel agora passa para a rede externa B e se registra no agente externo na rede B, que informa ao agente nativo o novo COA do nó móvel. Desse ponto em diante, o agente nativo redirecionará datagramas para a rede externa B. No que diz respeito ao correspondente, a mobilidade é transparente — datagramas são roteados por meio do mesmo agente nativo antes e depois de o nó móvel mudar de rede. Quanto ao agente nativo, não há qualquer ruptura no fluxo de datagramas — datagramas que chegam são repassados primeiro para a rede externa A; após a

mudança no COA, são repassados para a rede externa B. Mas o nó móvel verá um fluxo de datagramas interrompido ao se movimentar entre redes? Contanto que seja pequeno o tempo transcorrido entre o desligamento do nó móvel da rede A (ponto em que ele não pode mais receber datagramas via A) e sua conexão com a rede B (ponto em que registrará um novo COA no agente nativo da rede), poucos datagramas serão perdidos. Lembre-se de que dissemos, no Capítulo 3, que conexões fim a fim podem sofrer perda de datagramas por causa do congestionamento na rede. Por conseguinte, a perda ocasional de datagramas dentro de uma conexão quando um nó se move entre redes não é, de maneira alguma, um problema catastrófico. Se for preciso uma comunicação livre de perdas, mecanismos de camadas superiores recuperarão a perda de dados, quer elas resultem do congestionamento da rede ou da mobilidade do usuário.

Uma abordagem indireta de roteamento é utilizada no padrão IP móvel [RFC 5944], como discutiremos na Seção 6.6.

Roteamento direto para um nó móvel

A abordagem do roteamento indireto ilustrada na Figura 6.23 sofre de uma ineficiência conhecida como problema do roteamento triangular — datagramas endereçados ao nó móvel devem ser roteados primeiro para o agente nativo e em seguida para a rede externa, mesmo quando existir uma rota muito mais eficiente entre o correspondente e o nó móvel. No pior caso, imagine um usuário móvel que está visitando a rede externa de um colega. Os dois estão sentados lado a lado e trocando dados pela rede. Datagramas do correspondente (nesse caso, do colega do visitante) são roteados para o agente nativo do usuário móvel e, então, novamente de volta para a rede externa!

O roteamento direto supera a ineficiência do roteamento triangular, mas o faz à custa de complexidade adicional. Na abordagem do roteamento direto, um agente correspondente na rede do correspondente primeiro aprende o COA do nó móvel. Isso pode ser realizado fazendo o agente correspondente consultar o agente nativo, admitindo que (como é o caso no roteamento indireto) o nó móvel tem um valor atualizado para seu COA registrado no seu agente nativo. Também é possível que o próprio correspondente execute a função do agente correspondente, assim como um nó móvel poderia executar a função do agente externo. Tal situação é ilustrada como as etapas 1 e 2 na Figura 6.25. O agente correspondente, então, executa um túnel para os datagramas diretamente até o COA do nó móvel, de modo semelhante ao túnel executado pelo agente nativo, etapas 3 e 4 da Figura 6.25.

Embora supere o problema do roteamento triangular, o roteamento direto introduz dois importantes desafios adicionais:

- É preciso um **protocolo de localização de usuário móvel** para o agente correspondente consultar o agente nativo de modo a obter o COA do nó móvel (etapas 1 e 2 da Figura 6.25).
- Quando o nó móvel passa de uma rede externa para outra, como os dados são repassados, agora, para a nova rede? No caso do roteamento indireto, esse problema era facilmente resolvido atualizando-se o COA mantido pelo agente nativo. Todavia, com roteamento direto, o agente correspondente consulta o COA junto ao agente nativo apenas uma vez, no início da sessão. Assim, atualizar o COA do agente nativo, embora necessário, não será suficiente para resolver o problema do roteamento de dados para a nova rede externa do nó móvel.

Uma solução seria criar um novo protocolo para notificar a mudança de COA ao correspondente. Uma solução alternativa, que, como veremos, é adotada na prática em redes GSM, funciona da seguinte maneira: suponha que estão sendo repassados dados correntemente para o nó móvel na rede externa onde ele estava localizado quando a sessão foi iniciada (etapa 1 na Figura 6.26). O agente externo naquela rede externa onde o nó móvel foi encontrado pela primeira vez será denominado agente externo âncora. Quando o nó móvel passar para uma nova rede externa (etapa 2 na Figura 6.26), ele se registrará com o novo agente externo (etapa 3), o qual fornecerá ao agente externo âncora o novo COA do nó móvel (etapa 4). Quando o agente externo âncora receber um datagrama encapsulado para um nó móvel que já deixou a rede, ele então poderá reencapsular o datagrama e repassá-lo

para o nó móvel (etapa 5) usando o novo COA. Se, mais tarde, o nó móvel passar para mais alguma outra rede externa, o agente externo nessa nova rede visitada então contatará o agente externo âncora para estabelecer repasse para essa nova rede externa.

FIGURA 6.25 ROTEAMENTO DIRETO PARA UM USUÁRIO MÓVEL

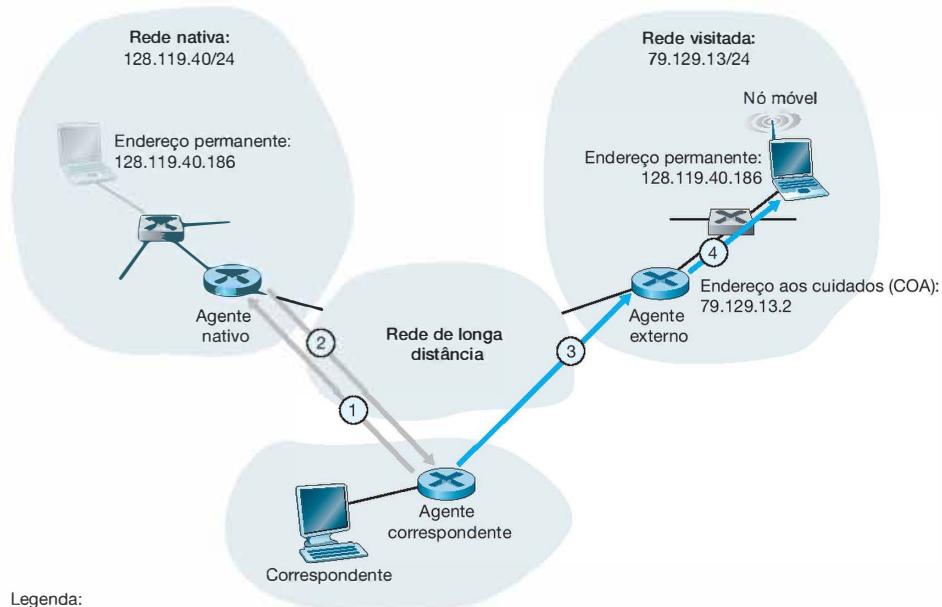
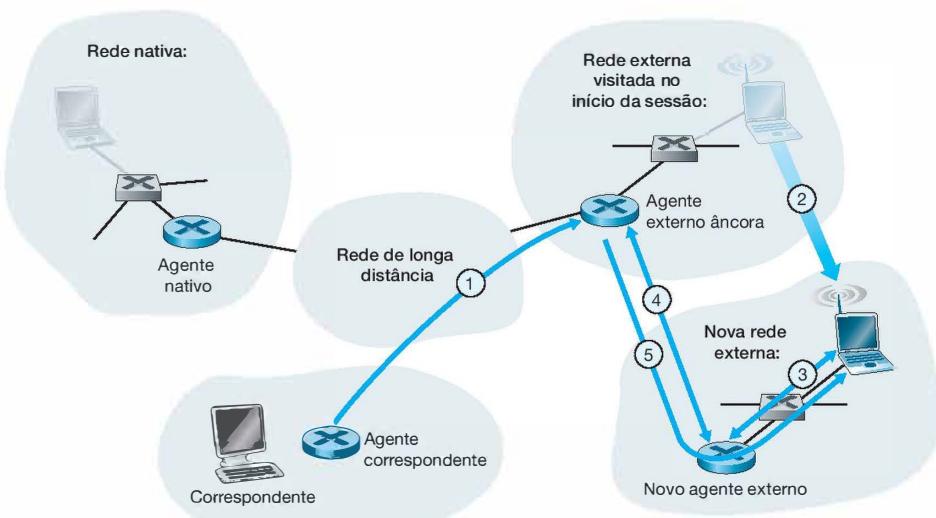



FIGURA 6.26 TRANSFERÊNCIA MÓVEL ENTRE REDES COM ROTEAMENTO DIRETO

6.6 IP MÓVEL

A arquitetura e os protocolos da Internet para suporte de mobilidade, conhecidos como IP móvel, estão definidos principalmente no RFC 5944 para IPv4. O IP móvel é um protocolo flexível que suporta muitos modos de operação diferentes (por exemplo, operação com ou sem um agente externo), várias maneiras para agentes e nós móveis descobrirem uns aos outros, utilização de um único COA ou de vários COAs e diversas formas de encapsulamento. Por isso, o IP móvel é um protocolo complexo, cuja descrição detalhada exigiria um livro inteiro; um desses livros é Perkins [1998b]. Aqui, nossa modesta meta é oferecer uma visão geral dos aspectos mais importantes do IP móvel e ilustrar sua utilização em alguns cenários comuns.

A arquitetura do IP móvel contém muitos dos elementos que acabamos de considerar, incluindo os conceitos de agentes nativos, agentes externos, endereços administrados e encapsulamento/desencapsulamento. O padrão atual [RFC 5944] especifica a utilização de roteamento indireto para o nó móvel.

O padrão IP móvel consiste em três partes principais:

- *Descoberta de agente.* O IP móvel define os protocolos utilizados por um agente nativo ou por um agente externo para anunciar seus serviços a nós móveis e protocolos para que os nós móveis solicitem os serviços de um agente externo ou nativo.
- *Registro no agente nativo.* O IP móvel define os protocolos usados pelo nó móvel e/ou agente externo para registrar e anular os registros de COAs no agente local de um nó móvel.
- *Roteamento indireto de datagramas.* O padrão também define a maneira pela qual datagramas são repassados para nós móveis por um agente nativo, incluindo regras para repassar datagramas, regras para manipular condições de erro e diversas formas de encapsulamento [RFC, 2003; RFC, 2004].

Considerações de segurança têm destaque em todo o padrão IP móvel. Por exemplo, a autenticação de um nó móvel é claramente necessária para impedir que um usuário mal-intencionado registre no agente nativo um falso endereço administrado, o que poderia fazer datagramas endereçados a um endereço IP serem direcionados ao usuário mal-intencionado. O IP móvel consegue segurança usando muitos dos mecanismos que examinaremos no Capítulo 8, portanto, não consideraremos a questão da segurança de endereços na discussão a seguir.

Descoberta de agente

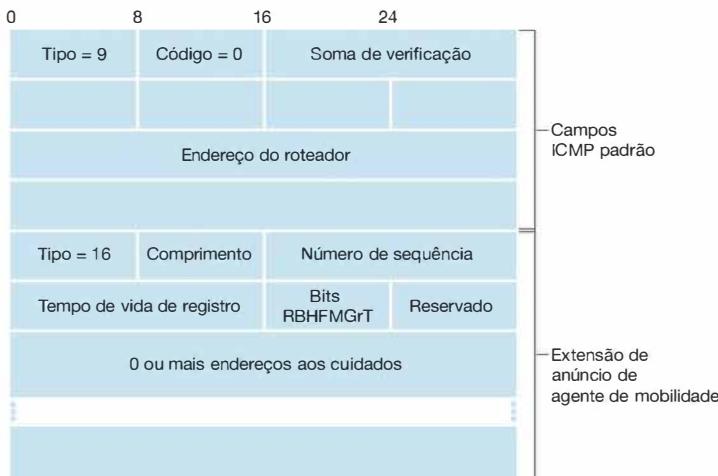
Um nó IP móvel que está chegando a uma nova rede, esteja ele se conectando a uma rede externa ou retornando à sua rede nativa, tem de aprender a identidade do correspondente externo ou do agente nativo. De fato, é a descoberta de um novo agente externo, com um novo endereço de rede, que permite à camada de rede em um nó móvel descobrir que ele passou para uma nova rede externa. Esse processo é conhecido como **descoberta de agente**. A descoberta de agente pode ser realizada de duas maneiras: via anúncio de agente ou via solicitação de agente

No caso do anúncio de agente, um agente externo ou nativo anuncia seus serviços usando uma extensão do protocolo de descoberta de roteador existente [RFC 1256]. O agente transmite periodicamente por difusão uma mensagem ICMP tendo 9 no campo de tipo (descoberta de roteador) em todos os enlaces aos quais está conectado. A mensagem de descoberta de roteador contém o endereço IP do roteador (isto é, o agente), e isso permite que um nó móvel descubra o endereço IP do agente. A mensagem de descoberta de roteador também contém uma extensão de anúncio de agente de mobilidade que guarda informações adicionais de que o nó móvel necessita. Entre os campos mais importantes na extensão estão os seguintes:

- *Bit do agente nativo (H).* Indica que o agente é um agente nativo para a rede na qual reside.
- *Bit de agente externo (F).* Indica que o agente é um agente externo para a rede na qual reside.
- *Bit de registro obrigatório (R).* Indica que um usuário móvel nessa rede *deverá* se registrar em um agente externo. Em particular, um usuário móvel não pode obter um endereço administrado na rede externa

(por exemplo, usando DHCP) e assumir a funcionalidade de agente externo para si mesmo sem se registrar no agente externo.

- *Bits de encapsulamento M, G.* Indicam se será utilizada uma forma de encapsulamento que não seja o encapsulamento IP em IP.
- *Campos de endereços aos cuidados (COA).* Uma lista de um ou mais endereços aos cuidados, fornecida pelo agente externo. No exemplo logo a seguir, o COA estará associado com o agente externo, que receberá datagramas enviados ao COA e então os repassará para o nó móvel adequado. O usuário móvel selecionará um desses endereços como seu COA ao se registrar no seu agente nativo.

A Figura 6.27 ilustra alguns dos principais campos na mensagem de anúncio de agente.

Com solicitação de agente, um nó móvel que quer conhecer agentes sem esperar para receber um anúncio de agente pode transmitir uma mensagem de solicitação em difusão, que é nada mais que uma mensagem ICMP cujo valor de tipo é 10. Ao receber a solicitação, um agente transmite um anúncio individual de agente diretamente ao nó móvel, que, então, procederá como se tivesse recebido um anúncio não solicitado.

Registro no agente nativo

Tão logo um nó móvel IP tenha recebido um COA, o endereço deve ser registrado no agente nativo, o que pode ser feito por meio do agente externo (que, então, registra o COA no agente nativo) ou diretamente pelo próprio nó móvel IP. A seguir, consideramos o primeiro caso. Há quatro etapas envolvidas.

1. Após o recebimento de um anúncio de agente externo, um nó móvel envia uma mensagem de registro IP móvel ao agente externo. A mensagem de registro é transportada dentro de um datagrama UDP e enviada à porta 434. A mensagem de registro leva um COA anunciado pelo agente externo, o endereço do agente nativo (HA), o endereço permanente do nó móvel (MA), o tempo de vida de registro requerido e uma identificação de registro de 64 bits. O tempo de vida de registro requerido é o número de segundos durante os quais o registro será válido. Se o registro não for renovado no agente nativo dentro do tempo de vida especificado, ele se tornará inválido. O identificador de registro age como um número de sequência e serve para combinar uma resposta de registro recebida com uma solicitação de registro, como discutiremos mais adiante.

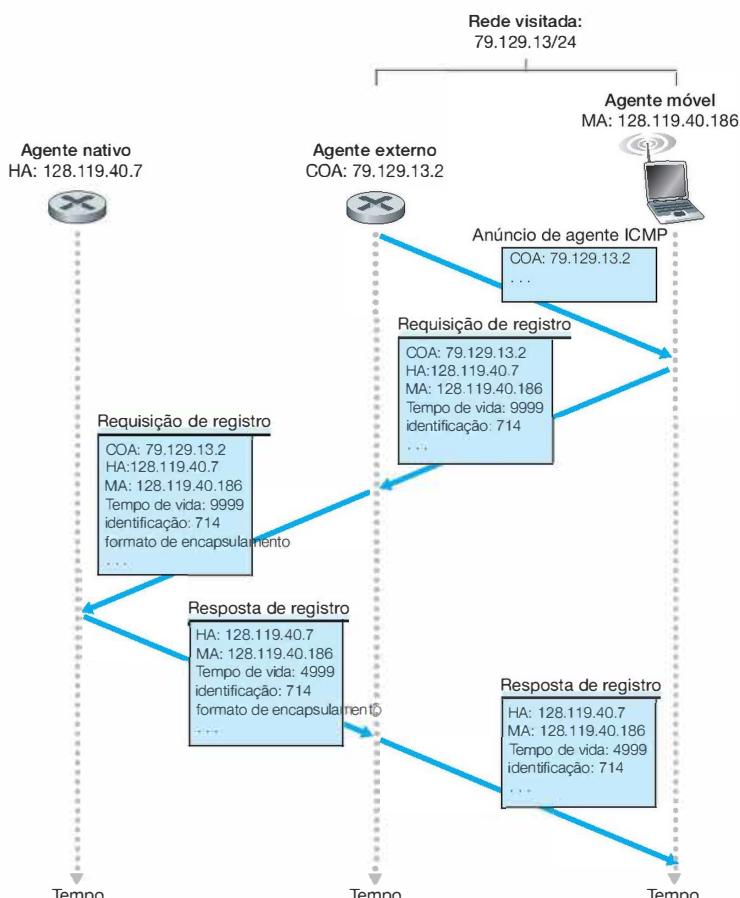



FIGURA 6.27 MENSAGEM ICMP DE DESCOBERTA DE ROTEADOR COM EXTENSÃO DE ANÚNCIO DE AGENTE DE MOBILIDADE

2. O agente externo recebe a mensagem de registro e registra o endereço IP permanente do nó móvel. Agora o agente externo sabe que deve procurar datagramas que contenham um datagrama encapsulado cujo endereço de destino combine com o endereço permanente do nó móvel. O agente externo, então, envia uma mensagem de registro do IP móvel (novamente, dentro de um datagrama UDP) à porta 434 do agente nativo. A mensagem contém o COA, o HA, o MA, o formato de encapsulamento requisitado, o tempo de vida de registro requisitado e a identificação do registro.
3. O agente nativo recebe a requisição de registro e verifica sua autenticidade e exatidão. O agente nativo vincula o endereço IP permanente do nó móvel ao COA; no futuro, datagramas que chegarem ao agente nativo e endereçados ao nó móvel serão encapsulados e enviados por túnel até o COA. O agente nativo envia uma resposta de registro IP móvel contendo o HA, o MA, o tempo de vida de registro vigente e a identificação de registro da solicitação que está sendo atendida com essa resposta.
4. O agente externo recebe a resposta de registro e então a repassa ao nó móvel.

Nesse ponto o registro está concluído e o nó móvel pode receber datagramas enviados a seu endereço permanente. A Figura 6.28 ilustra essas etapas. Note que o agente nativo especifica um tempo de vida menor do que aquele requisitado pelo nó móvel.

FIGURA 6.28 ANÚNCIO DE AGENTE E REGISTRO IP MÓVEL

Um agente externo não precisa anular explicitamente um registro de COA quando um nó móvel sai da rede. Isso ocorrerá automaticamente quando o nó móvel passar para uma nova rede (seja outra rede externa, seja sua rede nativa) e registrar um novo COA.

O padrão IP móvel permite muitos cenários e capacidades adicionais além das que acabamos de descrever. O leitor interessado deve consultar Perkins [1998b]; RFC [5944].

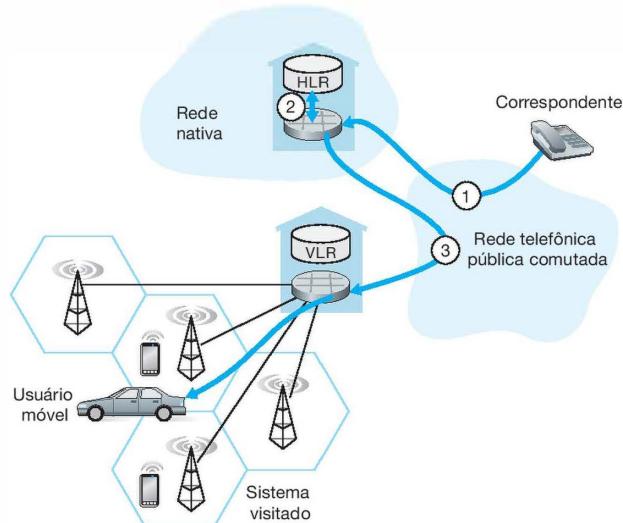
6.7 GERENCIAMENTO DE MOBILIDADE EM REDES CELULARES

Agora que acabamos de examinar como a mobilidade é gerenciada em redes IP, vamos voltar nossa atenção para redes cujo histórico de suporte à mobilidade é ainda mais longo — redes de telefonia celular. Enquanto na Seção 6.4 focalizamos o enlace sem fio do primeiro salto em redes celulares, aqui focalizaremos a mobilidade utilizando a arquitetura de rede celular GSM [Goodman, 1997; Mouly, 1992; Scourias, 2012; Kaaranen, 2001; Korhonen, 2003; Turner, 2012] como objeto de nosso estudo, visto que é uma tecnologia madura e amplamente disponibilizada. Como no caso do IP móvel, veremos que vários dos princípios fundamentais que identificamos da Seção 6.5 estão incorporados à arquitetura de rede do GSM.

Do mesmo modo que o IP móvel, o GSM adota uma abordagem de roteamento indireto (veja Seção 6.5.2), primeiro roteando a chamada do correspondente para a rede nativa do nó móvel e daí para a rede visitada. Em terminologia GSM, a rede nativa do nó móvel é denominada **rede pública terrestre móvel nativa (PLMN nativa).** Visto que o acrônimo PLMN é um pouco complicado, e sempre firmes na nossa decisão de evitar uma sopa de letrinhas, denominaremos a rede PLMN nativa GSM apenas **rede nativa**. A rede nativa é a operadora de celular da qual o usuário móvel é assinante (isto é, a operadora que cobra mensalmente do usuário pelo serviço celular). A PLMN visitada, que denominaremos **rede visitada**, é a rede na qual o nó móvel estiver residindo.

Como no caso do IP móvel, as responsabilidades das redes nativas e visitadas são bastante diferentes.

- A rede nativa mantém um banco de dados conhecido como **registro nativo de localização** (*home location register* — HLR), que contém o número permanente do telefone celular e as informações do perfil do assinante para cada assinante. O importante é que o HLR também contém informações sobre as localizações atuais desses assinantes. Isto é, se um usuário móvel estiver em trânsito pela rede celular de outra operadora, o HLR conterá informações suficientes para obter (por meio de um processo que descreveremos em breve) um endereço na rede visitada para o qual deverá ser roteada uma chamada ao usuário móvel. Como veremos, quando é feita uma chamada para um usuário móvel, um comutador especial na rede nativa, conhecido como **Central de Comutação para Portal de Serviços Móveis** (*Gateway Mobile Services Switching Center* — GMSC), é contatado por um correspondente. Mais uma vez, conforme nossa decisão de evitar a sopa de letrinhas, denominaremos a GMSC por um termo mais descritivo, uma **MSC nativa.**
- A rede visitada mantém um banco de dados conhecido como **registro de localização de visitantes** (*visitor location register* — VLR). O VLR contém um registro para cada usuário móvel que está *atualmente* na parte da rede atendida pelo VLR. Assim, os registros do VLR vêm e vão à medida que usuários entram e saem da rede. Um VLR normalmente está localizado juntamente com a central de comutação móvel (MSC) que coordena o estabelecimento de uma chamada de e para a rede visitada.

Na prática, a rede celular de uma operadora servirá como uma rede nativa para seus assinantes e como uma rede visitada para usuários móveis que são assinantes de outras operadoras de serviços celulares.

6.7.1 Roteando chamadas para um usuário móvel

Agora estamos prontos para descrever como é estabelecida uma chamada para um usuário GSM em uma rede visitada. A seguir, consideraremos um exemplo simples; cenários mais complexos são descritos em Mouly [1992]. As etapas, como ilustradas na Figura 6.29, são as seguintes:

1. O correspondente disca o número do telefone do usuário móvel. Esse número, em si, não se refere a uma determinada linha telefônica ou localização (afinal, o número do telefone é fixo, mas o usuário é móvel!). Os dígitos iniciais do número são suficientes para identificar globalmente a rede nativa do usuário móvel. A chamada é roteada desde o correspondente, passa através do PSTN e chega até a MSC na rede nativa do usuário móvel. Esse é o primeiro trecho da chamada.
2. A MSC nativa recebe a chamada e interroga o HLR para determinar a localização do usuário móvel. No caso mais simples, o HLR retorna o **número *roaming* da estação móvel** (*mobile station roaming number* — MSRN), que denominaremos **número de *roaming***. Note que esse número é diferente do número permanente do telefone móvel, que é associado com a rede nativa do usuário móvel. O número de *roaming* é efêmero: é designado temporariamente a um usuário móvel quando ele entra em uma rede visitada. O número de *roaming* desempenha um papel semelhante ao do endereço COA no IP móvel e, da mesma forma, é invisível para o correspondente e para o usuário móvel. Se o HLR não tiver o número de *roaming*, ele retornará ao endereço do VLR na rede visitada. Nesse caso (que não é mostrado na Figura 6.29), a MSC nativa precisará consultar o VLR para obter o número de *roaming* do nó móvel. Mas, antes de tudo, como o HLR obtém o número de *roaming* ou o endereço VLR? O que acontece a esses valores quando o usuário móvel passa para outra rede visitada? Em breve consideraremos essas perguntas importantes.
3. Dado o número de *roaming*, a MSC nativa estabelece o segundo trecho da chamada através da rede até a MSC na rede visitada. A chamada está concluída — foi roteada do correspondente até a MSC nativa e daí para a MSC visitada, e desta até a estação-base que atende ao usuário móvel.

Uma questão não resolvida na etapa 2 é como o HLR obtém informação sobre a localização do usuário móvel. Quando um telefone móvel é ligado ou entra em uma parte de uma rede visitada que é coberta por um novo VLR, ele deve se registrar na rede visitada. Isso é feito por meio da troca de mensagens de sinalização entre o usuário móvel e o VLR. O VLR visitado, por sua vez, envia uma mensagem de requisição de atualização de localização ao HLR do usuário móvel. Essa mensagem informa ao HLR o número de *roaming* no qual o usuário móvel pode ser contatado ou o endereço do VLR (que então pode ser consultado mais tarde para obter esse número). Como parte dessa troca, o VLR também obtém do HLR informações sobre o assinante do usuário móvel e determina quais serviços (se houver algum) devem ser prestados a ele pela rede visitada.

FIGURA 6.29 ESTABELECENDO UMA CHAMADA PARA UM USUÁRIO MÓVEL: ROTEAMENTO INDIRETO

6.7.2 Transferências (*handoffs*) em GSM

Uma **transferência** (*handoff*) ocorre quando uma estação móvel muda sua associação de uma estação-base para outra durante uma chamada. Como ilustra a Figura 6.30, uma chamada de telefone móvel é roteada de início (antes da transferência) para o usuário móvel por meio de uma estação-base (a qual denominaremos antiga estação-base) e, após, por meio de outra estação-base (a qual denominaremos nova estação-base). Note que uma transferência entre estações-base resulta não apenas em transferência/recepção de/para um telefone móvel e uma nova estação-base, mas também no redirecionamento da chamada em curso de um ponto de comutação dentro da rede para a nova estação-base. Vamos admitir inicialmente que as estações-base antiga e nova compartilham a mesma MSC e que o redirecionamento ocorre nessa MSC.

Há diversas razões possíveis para ocorrer transferência, incluindo (1) o sinal entre a estação-base corrente e o usuário móvel pode ter-se deteriorado a tal ponto que a chamada corre perigo de "cair" e (2) uma célula pode ter ficado sobrecarregada, manipulando grande número de chamadas. Esse congestionamento pode ser aliviado transferindo usuários móveis para células próximas, menos congestionadas.

Enquanto está associado com uma estação-base, um usuário móvel mede periodicamente a potência de um sinal de sinalização emitido por sua estação-base corrente, bem como de sinais de sinalização emitidos por estações-base próximas que ele pode "ouvir". Essas medições são passadas uma ou duas vezes por segundo para a estação-base corrente do usuário móvel. A transferência em GSM é iniciada pela estação-base antiga com base nessas medições, nas cargas correntes de usuários móveis em células próximas e outros fatores [Mouly, 1992]. O padrão GSM não determina o algoritmo específico a ser utilizado por uma estação-base para decidir se realiza ou não uma transferência.

A Figura 6.31 ilustra as etapas envolvidas quando uma estação-base decide transferir um usuário móvel:

1. A antiga estação-base (BS) informa à MSC visitada que deve ser feita uma transferência e a BS (ou possível conjunto de BSs) para a qual o usuário móvel deve ser transferido.
2. A MSC visitada inicia o estabelecimento do caminho até a nova BS, alocando os recursos necessários para carregar a chamada redirecionada e sinalizando à nova BS que uma transferência está prestes a ocorrer.
3. A nova BS reserva e ativa um canal de rádio para ser utilizado pelo usuário móvel.
4. A nova BS devolve um sinal à MSC visitada e à antiga BS, indicando que foi estabelecido um caminho entre a MSC visitada e a nova BS e que o usuário móvel deve ser informado da transferência iminente. A nova BS provê todas as informações de que o usuário móvel necessitará para se associar com a nova BS.
5. O usuário móvel é informado de que deve realizar uma transferência. Note que, até esse ponto, ele está totalmente desavisado de que a rede estava preparando o terreno para uma transferência (por exemplo, reservando um canal na nova BS e alocando um caminho entre a MSC visitada e a nova BS).
6. O usuário móvel e a nova BS trocam uma ou mais mensagens para ativar totalmente o novo canal na nova BS.

FIGURA 6.30 CENÁRIO DE TRANSFERÊNCIA ENTRE ESTAÇÕES-BASE QUE TÊM UMA MSC EM COMUM

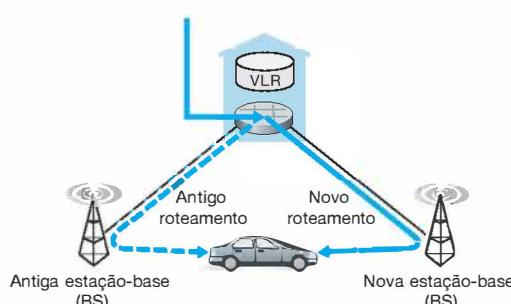
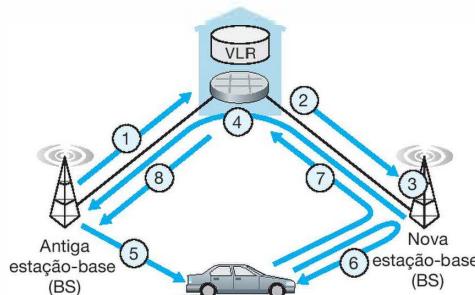



FIGURA 6.31 ETAPAS DA EXECUÇÃO DE UMA TRANSFERÊNCIA ENTRE ESTAÇÕES-BASE QUE TÊM UMA MSC EM COMUM

7. O móvel envia à nova BS uma mensagem de conclusão de transferência que é repassada para a MSC visitada. Então, a MSC visitada redireciona a chamada em curso para o usuário móvel, por meio da nova BS.
8. Os recursos reservados ao longo do caminho até a antiga BS são liberados.

Vamos concluir nossa discussão sobre transferência considerando o que acontece quando o usuário móvel passa para uma BS associada com uma MSC *diferente* da antiga BS e o que acontece quando essa transferência entre MSCs ocorre mais de uma vez. Como ilustra a Figura 6.32, o GSM define a noção de uma MSC âncora. A MSC âncora é a MSC visitada pelo usuário móvel logo no início de uma chamada; assim, ela não muda durante a chamada. Por toda a duração da chamada e independentemente do número de transferências entre MSCs realizadas pelo usuário móvel, a chamada é roteada desde a MSC nativa até a âncora e, então, desta até a MSC visitada, onde o usuário móvel está localizado no momento. Quando um usuário móvel passa da área de cobertura de uma MSC para a área de cobertura de outra, a chamada em curso é redirecionada desde a MSC âncora até a nova MSC visitada, que contém a nova estação-base. Assim, durante o tempo todo há, no máximo, três MSCs

FIGURA 6.32 REDIRECIONAMENTO VIA A MSC ÂNCORA

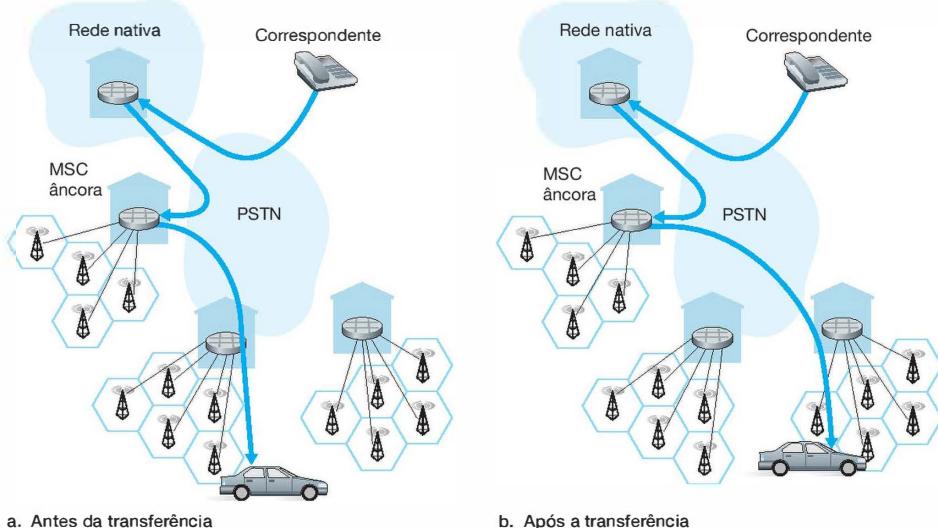



TABELA 6.2 SEMELHANÇAS ENTRE A MOBILIDADE EM IP MÓVEL E EM GSM

Elemento do GSM	Comentário sobre o elemento do GSM	Elemento do IP móvel
Sistema nativo	Rede à qual pertence o número de telefone permanente do usuário.	Rede nativa
Central de comutação de unidade móvel ou simplesmente MSC nativa, Registro nativo de localização (HLR)	MSC nativa: ponto de contato para obter endereço roteável de usuário móvel. HLR: banco de dados no sistema nativo que contém número de telefone permanente, informações de perfil, localização corrente de usuário móvel, informações de assinatura.	Agente nativo
Sistema visitado	Rede, exceto o sistema nativo, onde o usuário móvel está residindo.	Rede visitada
Central de serviços de comutação de unidade móvel visitada, Registro de Localização de Visitante (VLR)	MSC visitada: responsável por estabelecer chamadas de/para nós móveis em células associadas com MSC. VLR: registro temporário em banco de dados em sistema visitado, contendo informações de assinatura para cada usuário móvel visitado.	Agente externo
Número de *roaming* de estação móvel (MSRN) ou simplesmente número de roaming	Endereço roteável para segmento de chamada telefônica entre MSC nativa e MSC visitada, que não é visível nem para o usuário móvel nem para o correspondente.	Endereço administrado

(a nativa, a âncora e a visitada) entre o correspondente e o usuário móvel. A Figura 6.32 ilustra o roteamento de uma chamada entre as MSCs visitadas por um usuário móvel.

Em vez de manter um único salto desde a MSC âncora até a MSC corrente, uma técnica alternativa seria encadear as MSCs visitadas pelo móvel, fazendo uma MSC antiga transmitir a chamada em curso até a nova MSC cada vez que o móvel passa para uma nova MSC. Esse encadeamento de MSC pode de fato ocorrer em redes celulares IS-41, com uma etapa opcional de minimização de caminho para remover MSCs entre a âncora e a nova visitada [Lin, 2001].

Vamos encerrar nossa discussão sobre o gerenciamento da mobilidade GSM fazendo uma comparação entre gerenciamento de mobilidade em GSM e em IP Móvel. A comparação apresentada na Tabela 6.2 indica que, embora redes IP e celulares sejam em essência diferentes em muitos aspectos, compartilham um número surpreendente de elementos funcionais comuns e abordagens gerais para o tratamento da mobilidade.

6.8 SEM FIO E MOBILIDADE: IMPACTO SOBRE PROTOCOLOS DE CAMADAS SUPERIORES

Neste capítulo, vimos que redes sem fio são significativamente diferentes de suas contrapartes cabeadas tanto na camada de enlace (como resultado de características de canais sem fio como desvanecimento, propagação multivias e terminais ocultos) quanto na camada de rede (como resultado de usuários móveis que mudam seus pontos de conexão com a rede). Mas há diferenças importantes nas camadas de transporte e de aplicação? É tentador pensar que essas diferenças seriam pequenas, visto que a camada de rede provê o mesmo modelo de serviço de entrega de melhor esforço às camadas superiores tanto em redes cabeadas quanto em redes sem fio. De modo semelhante, se protocolos como TCP ou UDP são usados para oferecer serviços da camada de transporte a aplicações tanto em redes cabeadas como em redes sem fio, então a camada de aplicação também deve permanecer inalterada. Nossa intuição está certa em um sentido — TCP e UDP podem operar, e de fato operam, em redes com enlaces sem fio. Por outro lado, protocolos de transporte em geral e o TCP em particular às vezes podem ter desempenhos muito diferentes em redes cabeadas e em redes sem fio, e é neste particular, em termos de desempenho, que as diferenças se manifestam. Vejamos por quê.

Lembre-se de que o TCP retransmite um segmento que é perdido ou corrompido no caminho entre remetente e destinatário. No caso de usuários móveis, a perda pode resultar de congestionamento de rede (estouro de buffer de roteador) ou de transferência (por exemplo, de atrasos no redirecionamento de segmentos para um novo ponto de conexão do usuário à rede). Em todos os casos, o ACK do destinatário ao remetente do TCP indica apenas que um segmento não foi recebido intacto; o remetente não sabe se o segmento foi perdido por congestionamento, durante a transferência, ou por erros de bits detectados. Em todos os casos, a resposta do remetente é a mesma — retransmitir o segmento. A resposta do controle de congestionamento do TCP *também* é a mesma em todos os casos — o TCP reduz sua janela de congestionamento, como discutimos na Seção 3.7. Reduzindo de modo incondicional sua janela de congestionamento, o TCP admite implicitamente que a perda de segmento resulta de congestionamento e não de corrupção ou transferência. Vimos na Seção 6.2 que erros de bits são muito mais comuns em redes sem fio do que nas cabeadas. Quando ocorrem esses erros de bits ou quando há perda na transferência, na realidade não há razão alguma para que o remetente TCP reduza sua janela de congestionamento (reduzindo, assim, sua taxa de envio). Na verdade, é bem possível que os buffers de roteador estejam vazios e que pacotes estejam fluindo ao longo de caminhos fim a fim desimpedidos, sem congestionamento.

Entre o início e meados da década de 1990, pesquisadores perceberam que, dadas as altas taxas de erros de bits em enlaces sem fio e a possibilidade de perdas pela transferência de usuários, a resposta do controle de congestionamento do TCP poderia ser problemática em um ambiente sem fio. Há duas classes gerais de abordagens possíveis para tratar esse problema:

- *Recuperação local.* Os protocolos de recuperação local recuperam erros de bits quando e onde (por exemplo, no enlace sem fio) eles ocorrem (por exemplo, o protocolo ARQ 802.11, que estudamos na Seção 6.3, ou técnicas mais sofisticadas que utilizam ARQ e também FEC [Ayanoglu, 1995]).
- *Remetente TCP ciente de enlaces sem fio.* Em técnicas de recuperação locais, o remetente TCP fica completamente desavisado de que seus segmentos estão atravessando um enlace sem fio. Uma técnica alternativa é o remetente e o destinatário ficarem cientes da existência de um enlace sem fio, para distinguir entre perdas por congestionamento na rede cabeada e corrupção/perdas no enlace sem fio, e invocar o controle de congestionamento somente em resposta a perdas por congestionamento na rede cabeada. Balakrishnan [1997] investiga vários tipos de TCP, supondo que sistemas finais possam fazer essa distinção. Liu [2003] investiga técnicas para distinguir entre perdas nos segmentos cabeado e sem fio de um caminho fim a fim.
- *Técnicas de conexão dividida.* Nesta técnica de conexão dividida [Bakre, 1995], a conexão fim a fim entre o usuário móvel e o outro ponto terminal é dividida em duas conexões da camada de transporte: uma do hospedeiro móvel ao ponto de acesso sem fio, e uma do ponto de acesso sem fio ao outro ponto terminal de comunicação (admitiremos, aqui, um usuário cabeado). A conexão fim a fim é, então, formada por uma concatenação de uma parte sem fio e uma parte cabeada. A camada de transporte sobre um segmento sem fio pode ser uma conexão-padrão TCP [Bakre, 1995], ou principalmente um protocolo de recuperação de erro personalizado em cima do UDP. Yavatkar [1994] analisa o uso de um protocolo de repetição seletiva da camada de transporte por uma conexão sem fio. As medidas relatadas em Wei [2006] indicam que conexões TCP divididas são bastante usadas em redes de dados celulares, e que aperfeiçoamentos significativos podem ser feitos com o uso dessas conexões.

Aqui, nosso tratamento do TCP em enlaces sem fio foi necessariamente breve. Estudos mais aprofundados dos desafios e soluções do TCP nessas redes podem ser encontrados em Hanabali [2005]; Leung [2006]. Aconselhamos o leitor a consultar as referências se quiser mais detalhes sobre essa área de pesquisa em curso.

Agora que já consideramos protocolos de camada de transporte, vamos analisar em seguida o efeito do sem fio e da mobilidade sobre protocolos da camada de aplicação. Uma consideração importante aqui é que enlaces sem fio muitas vezes têm larguras de banda relativamente baixas, como vimos na Figura 6.2. Por conseguinte, aplicações que operam por enlaces sem fio, em particular por enlaces celulares sem fio, devem

tratar a largura de banda como uma mercadoria escassa. Por exemplo, um servidor Web que serve conteúdo a um navegador Web que está rodando em um telefone 3G talvez não consiga prover o mesmo conteúdo rico em imagens que oferece a um navegador que está rodando sobre uma conexão cabeada. Embora enlaces sem fio proponham desafios na camada de aplicação, a mobilidade que eles criam também torna possível um rico conjunto de aplicações cientes de localização e de contexto [Chen, 2000; Baldauf, 2007]. Em termos mais gerais, redes sem fio e redes móveis desempenharão um papel fundamental na concretização dos ambientes de computação onipresentes do futuro [Weiser, 1991]. É justo dizer que vimos somente a ponta do iceberg quando se trata do impacto de redes sem fio e móveis sobre aplicações em rede e seus protocolos!

6.9 RESUMO

Redes sem fio e móveis revolucionaram a telefonia e também estão causando um impacto cada vez mais profundo no mundo das redes de computadores. Com o acesso à infraestrutura da rede global que oferecem — desimpedido, a qualquer hora, em qualquer lugar — estão não só aumentando a onipresença do acesso a redes, mas também habilitando um novo conjunto muito interessante de serviços dependentes de localização. Dada a crescente importância das redes sem fio e móveis, este capítulo focalizou os princípios, as tecnologias de enlace e as arquiteturas de rede para suportar comunicações sem fio e móveis.

Iniciamos o capítulo com uma introdução às redes sem fio e móveis, traçando uma importante distinção entre os desafios propostos pela natureza *sem fio* dos enlaces de comunicação desse tipo de rede, e pela *mobilidade* que permitem. Isso nos possibilitou isolar, identificar e dominar melhor os conceitos fundamentais em cada área. Focalizamos primeiro a comunicação sem fio, considerando as características de um enlace sem fio na Seção 6.2. Nas seções 6.3 e 6.4 examinamos os aspectos de camada de enlace do padrão IEEE 802.11 (Wi-Fi) para LANs sem fio, duas redes pessoais IEEE 802.15 (Bluetooth e Zigbee) e acesso à Internet pela rede celular 3G e 4G. Depois, voltamos nossa atenção para a questão da mobilidade. Na Seção 6.5, identificamos diversas formas de mobilidade e verificamos que há pontos nesse espectro que propõem desafios diferentes e admitem soluções diferentes. Consideramos os problemas de localização e roteamento para um usuário móvel, bem como técnicas para transferir o usuário móvel que passa dinamicamente de um ponto de conexão com a rede para outro. Examinamos como essas questões foram abordadas no padrão IP móvel e em GSM nas seções 6.6 e 6.7, respectivamente. Por fim, na Seção 6.8, consideramos o impacto causado por enlaces sem fio e pela mobilidade sobre protocolos de camada de transporte e aplicações em rede.

Embora tenhamos dedicado um capítulo inteiro ao estudo de redes sem fio e redes móveis, seria preciso todo um livro (ou mais) para explorar completamente esse campo tão animador e que está se expandindo tão depressa. Aconselhamos o leitor a se aprofundar mais nesse campo consultando as muitas referências fornecidas neste capítulo.

EXERCÍCIOS DE FIXAÇÃO E PERGUNTAS

Questões de revisão do Capítulo 6

SEÇÃO 6.1

R1. O que significa para uma rede sem fio estar operando no "modo de infraestrutura"? Se a rede não estiver nesse modo, em qual modo ela está e qual é a diferença entre esse modo de operação e o de infraestrutura?

R2. Quais são os quatro tipos de redes sem fio identificadas em nossa taxonomia na Seção 6.1? Quais desses tipos de rede sem fio você usou?

SEÇÃO 6.2

R3. Quais são as diferenças entre os seguintes tipos de falhas no canal sem fio: atenuação de percurso, propagação multivias, interferência de outras fontes?

R4. Um nó móvel se distancia cada vez mais de uma estação-base. Quais são as duas atitudes que uma estação--base poderia tomar para garantir que a probabilidade de perda de um quadro transmitido não aumente?

SEÇÕES 6.3-6.4

R5. Descreva o papel dos quadros de sinalização em 802.11.

R6. Verdadeiro ou falso: antes de uma estação 802.11 transmitir um quadro de dados, ela deve primeiro enviar um quadro RTS e receber um quadro CTS correspondente.

R7. Por que são usados reconhecimentos em 802.11, mas não em Ethernet cabeada?

R8. Verdadeiro ou falso: Ethernet e 802.11 usam a mesma estrutura de quadro.

R9. Descreva como funciona o patamar RTS.

R10. Suponha que os quadros RTS e CTS IEEE 802.11 fossem tão longos quanto os padronizados DATA e ACK. Haveria alguma vantagem em usar os quadros CTS e RTS? Por quê?

R11. A Seção 6.3.4 discute mobilidade 802.11, na qual uma estação sem fio passa de um BSS para outro dentro da mesma sub-rede. Quando os APs estão interconectados com um comutador, um AP pode precisar enviar um quadro com um endereço MAC fingido para fazer o comutador transmitir quadros adequadamente. Por quê?

R12. Quais são as diferenças entre o dispositivo mestre em uma rede Bluetooth e uma estação-base em uma rede 802.11?

R13. O que significa um superquadro no padrão Zigbee 802.15.4?

R14. Qual é a função do "núcleo da rede" na arquitetura de dados celular 3G?

R15. Qual é a função do RNC na arquitetura da rede de dados celular 3G? Que função o RNC desempenha na rede celular de voz?

SEÇÕES 6.5-6.6

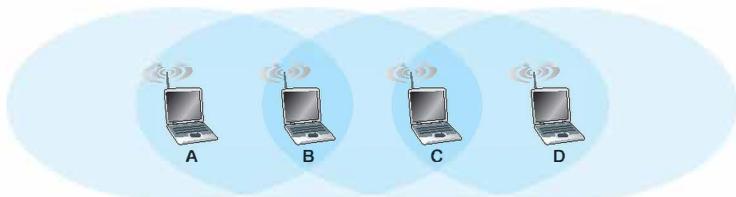
R16. Se um nó tem uma conexão sem fio à Internet, ele precisa ser móvel? Explique. Suponha que um usuário portando um notebook ande com ele pela casa, e sempre acesse a Internet por meio do mesmo ponto de acesso. Em relação à rede, este é usuário móvel? Explique.

R17. Qual é a diferença entre um endereço permanente e um endereço aos cuidados (COA)? Quem determina o endereço aos cuidados?

R18. Considere uma conexão TCP através de um IP móvel. Falso ou verdadeiro: a fase da conexão TCP entre o correspondente e o hospedeiro móvel percorre a rede doméstica móvel, mas a fase de transferência de dados está diretamente entre o correspondente e o hospedeiro móvel, pulando a rede doméstica.

SEÇÃO 6.7

R19. Quais são os objetivos do HLR e VLR nas redes GSM? Quais elementos de IP móvel são semelhantes ao HLR e ao VLR?

R20. Qual é o papel da MSC âncora em redes GSM?

SEÇÃO 6.8

R21. Quais são os três métodos que podem ser realizados para evitar que um único enlace sem fio reduza o desempenho de uma conexão TCP fim a fim da camada de transporte?

PROBLEMAS

P1. Considere o exemplo do remetente CDMA único na Figura 6.5. Qual seria a saída do remetente (para os 2 bits de dados mostrados) se o código do remetente CDMA fosse (1, −1, 1, −1, 1, −1, 1, −1)?

P2. Considere o remetente 2 na Figura 6.6. Qual é a saída do remetente para o canal (antes de ser adicionada ao sinal vindo do remetente 1), $Z^2_{i,m}$?

P3. Suponha que o receptor na Figura 6.6 queira receber os dados que estão sendo enviados pelo remetente 2. Mostre (por cálculo) que o receptor pode, na verdade, recuperar dados do remetente 2 do sinal agregado do canal usando o código do remetente 2.

P4. Para o exemplo sobre dois remetentes, dois destinatários, dê um exemplo de dois códigos CDMA contendo os valores 1 e −1, que não permitem que dois destinatários extraiam os bits originais transmitidos por dois remetentes CDMA.

P5. Suponha que dois ISPs fornecem acesso Wi-Fi em um determinado local, e que cada um deles opera seu próprio AP e tem seu próprio bloco de endereços IP.

a. Suponha ainda mais, que, por acidente, cada ISP configurou seu AP para operar no canal 11. O protocolo 802.11 falhará totalmente nessa situação? Discuta o que acontece quando duas estações, cada uma associada com um ISP diferente, tentam transmitir ao mesmo tempo.

b. Agora suponha que um AP opera no canal 1 e outro no canal 11. Como você mudaria suas respostas?

P6. Na etapa 4 do protocolo CSMA/CA, uma estação que transmite um quadro com sucesso inicia o protocolo CSMA/CA para um segundo quadro na etapa 2, e não na 1. Quais seriam as razões que os projetistas do CSMA/CA provavelmente tinham em mente para fazer essa estação não transmitir o segundo quadro de imediato (se o canal fosse percebido como ocioso)?

P7. Suponha que uma estação 802.11b seja configurada para sempre reservar o canal com a sequência RTS/CTS. Imagine que essa estação de repente queira transmitir 1.000 bytes de dados e que todas as outras estações estão ociosas nesse momento. Calcule o tempo requerido para transmitir o quadro e receber o reconhecimento como uma função de SIFS e DIFS, ignorando atraso de propagação e admitindo que não haja erros de bits.

P8. Considere o cenário mostrado na Figura 6.33, no qual existem quatro nós sem fios, A, B, C e D. A cobertura de rádio dos quatro nós é mostrada pelas formas ovais mais escuras; todos os nós compartilham a mesma frequência. Quando A transmite, ele pode ser ouvido/recebido por B; quando B transmite, ele só pode ser ouvido/recebido por A e C; quando C transmite, B e D podem ouvir/receber de C; quando D transmite, somente C pode ouvir/receber de D.

FIGURA 6.33 CENÁRIO PARA O PROBLEMA P8

Agora suponha que cada nó possua um estoque infinito de mensagens que ele queira enviar para os outros nós. Se o destinatário da mensagem não for um vizinho imediato, então a mensagem deve ser retransmitida. Por exemplo, se A quer enviar para D, uma mensagem de A deve ser primeiro enviada a B, que, então, envia a mensagem a C, e este a D. O tempo é dividido em intervalos, com um tempo de transmissão de mensagem de

exatamente um intervalo de tempo, como em um *slotted* Aloha, por exemplo. Durante um intervalo, um nó pode fazer uma das seguintes opções: (*i*) enviar uma mensagem; (*ii*) receber uma mensagem (se, exatamente, uma mensagem estiver sendo enviada a ele), (*iii*) permanecer silencioso. Como sempre, se um nó ouvir duas ou mais transmissões simultâneas, ocorrerá uma colisão e nenhuma das mensagens transmitidas é recebida com sucesso. Você pode admitir aqui que não existem erros de bits e, dessa forma, se uma mensagem for enviada, ela será recebida corretamente pelos que estão dentro do raio de transmissão do emissor.

a. Suponha que um controlador onisciente (ou seja, que sabe o estado de cada nó na rede) possa comandar cada nó a fazer o que ele (o controlador onisciente) quiser, isto é, enviar uma mensagem, receber uma mensagem, ou permanecer silencioso. Dado esse controlador onisciente, qual é a taxa máxima à qual uma mensagem de dados pode ser transferida de C para A, sabendo que não existem outras mensagens entre nenhuma outra dupla remetente/destinatária?

b. Suponha que A envie uma mensagem a B, e D envie uma mensagem a C. Qual é a taxa máxima combinada à qual as mensagens de dados podem fluir de A a B e de D a C?

c. Considere agora que A envie uma mensagem a B, e C envie uma mensagem a D. Qual é a taxa máxima combinada à qual as mensagens de dados podem fluir de A a B e de C a D?

d. Suponha agora que os enlaces sem fio sejam substituídos por enlaces cabeados. Repita as questões de "a" a "c" neste cenário cabeado.

e. Agora imagine que estamos de novo em um cenário sem fio e que para cada mensagem de dados enviada do remetente ao destinatário, este envie de volta uma mensagem ACK para o remetente (como no TCP, por exemplo). Suponha também que cada mensagem ACK possua um intervalo. Repita as questões de "a" a "c" para este cenário.

P9. Descreva o formato do quadro Bluetooth 802.15.1. Você precisará de uma leitura complementar para encontrar essa informação. Existe algo no formato do quadro que basicamente limite o número de nós ativos para oito em uma rede 802.15.1? Explique.

P10. Considere o seguinte cenário WiMAX ideal. O subquadro de descendente (veja Figura 6.17) é dividido em intervalos de tempo, com N intervalos descendentes por subquadro, e todos os intervalos de tempo têm o mesmo comprimento. Existem quatro nós, A, B, C e D, alcançáveis da estação-base a taxas de 10 Mbits/s, 5 Mbits/s, 2,5 Mbits/s e 1 Mbit/s, respectivamente no canal descendente. A estação-base possui infinitos dados para enviar a cada nó e pode enviar para qualquer um dos quatro nós durante qualquer intervalo de tempo no subquadro descendente.

a. Qual é a taxa máxima à qual a estação-base pode enviar aos nós, admitindo que ela pode enviar a qualquer nó de sua escolha durante cada intervalo de tempo? Sua solução é justa? Explique e defina o que você quis dizer com "justo".

b. Se há requisito de equidade que todos os nós devem receber uma quantidade igual de dados durante cada quadro de *downstream*, qual é a taxa média de transmissão pela estação-base (para todos os nós) durante o subquadro de *downstream*? Explique como você chegou a essa resposta.

c. Suponha que, como critério de equidade, qualquer nó possa receber, no máximo, duas vezes tantos dados quanto qualquer outro nó durante o subquadro. Qual é a taxa média de transmissão pela estação-base (para todos os nós) durante o subquadro de *downstream*? Explique como você chegou a esta resposta.

P11. Na Seção 6.5, uma solução proposta que permitia que usuários móveis mantivessem seu endereço IP à medida que transitavam entre redes externas era fazer uma rede externa anunciar ao usuário móvel uma rota altamente específica e usar a infraestrutura de roteamento existente para propagar essa informação por toda a rede. Uma das preocupações que identificamos foi a escalabilidade. Suponha que, quando um usuário móvel passa de uma rede para outra, a nova rede externa anuncie uma rota específica para o usuário móvel e a antiga rede externa retire sua rota. Considere como informações de roteamento se propagam em um algoritmo vetor de distâncias (em particular para o caso de roteamento interdomínios entre redes que abrangem o globo terrestre).

a. Outros roteadores conseguirão rotear datagramas imediatamente para a nova rede externa tão logo essa rede comece a anunciar sua rota?

b. É possível que roteadores diferentes acreditem que redes externas diferentes contenham o usuário móvel?

c. Discuta a escala temporal segundo a qual outros roteadores na rede finalmente aprenderão o caminho até os usuários móveis.

P12. Suponha que o correspondente na Figura 6.22 fosse móvel. Faça um desenho esquemático da infraestrutura adicional de camada de rede que seria necessária para rotear o datagrama do usuário móvel original até o correspondente (que agora é móvel). Mostre a estrutura do(s) datagrama(s) entre o usuário móvel original e o correspondente (agora móvel), como na Figura 6.23.

P13. Em IP móvel, que efeito terá a mobilidade sobre atrasos fim a fim de datagramas entre a fonte e o destino?

P14. Considere o exemplo de encadeamento discutido no final da Seção 6.7.2. Suponha que um usuário móvel visite as redes externas A, B e C, e que um correspondente inicie uma conexão com o usuário móvel enquanto este reside na rede externa A. Relacione a sequência de mensagens entre agentes externos e entre agentes externos e o agente nativo, enquanto o usuário passa da rede A para a rede B e para a rede C. Em seguida, suponha que não é executado encadeamento e que as mudanças no endereço administrado do usuário móvel devem ser notificadas explicitamente ao correspondente (bem como ao agente nativo). Relacione a sequência de mensagens que seria necessário trocar nesse segundo cenário.

P15. Considere dois nós móveis em uma rede externa que tem um agente externo. É possível que esses nós utilizem o mesmo endereço aos cuidados em IP móvel? Explique sua resposta.

P16. Quando discutimos como o VLR atualizava o HLR com informações sobre a localização corrente de usuários móveis, quais eram as vantagens e as desvantagens de fornecer ao HLR o MSRN em vez do endereço do VLR?

WIRESHARK LAB

No site de apoio do livro você encontrará um Wireshark Lab, em inglês, para este capítulo, que captura e estuda os quadros 802.11 trocados entre um notebook sem fio e um ponto de acesso.

ENTREVISTA

Deborah Estrin

Deborah Estrin é professora de ciência da computação na UCLA, Jon Postel Chair in Computer Networks, diretora do Center for Embedded Networked Sensing (CENS), e co-fundadora da organização sem fins lucrativos openmhealth.org. Ela recebeu doutorado (1985) em ciência da computação pelo MIT e bacharelado (1980) pela Universidade da Califórnia em Berkeley. O estudo inicial de Estrin foi voltado para o projeto de protocolos de rede, incluindo roteamento de transmissão para um grupo (*multicast*) e interdomínio. Em 2002, fundou o NSF-funded Science and Technology Center, CENS (http://cens.ucla.edu), para desenvolver e explorar tecnologias e aplicações de monitoração ambiental. Hoje, Estrin e seus colaboradores estão desenvolvendo sistemas de sensores participativos, aproveitando a programabilidade, proximidade e difusão dos telefones móveis; os principais contextos de implementação são saúde móvel (http://openmhealth.org), coleta de dados da comunidade e educação STEM (http://mobilizingcs.org). A professora Estrin é membro eleito da American Academy of Arts and Sciences (2007) e da National Academy of Engineering (2009). É *fellow* do IEEE, ACM e AAAS. Foi selecionada como a primeira ACM-W Athena Lecturer (2006), recebeu o prêmio Women of Vision Award for Innovation (2007) pelo Anita Borg Institute, entrou para o *hall* da fama do WITI (2008) e recebeu o prêmio Doctor Honoris Causa do EPFL (2008) e da Uppsala University (2011).

Por favor, descreva alguns dos projetos mais interessantes em que trabalhou durante sua carreira. Quais foram os maiores desafios?

Em meados da década de 1990 na USC e ISI, tive a sorte de trabalhar com pessoas como Steve Deering, Mark Handley e Van Jacobson no projeto de protocolos de roteamento para transmissão para grupos (em particular, PIM). Tentei fazer muitas das lições de projeto arquitetônico desde a transmissão para um grupo no projeto de vetores de monitoramento ecológico, onde, pela primeira vez, comecei a levar a sério as aplicações e a pesquisa multidisciplinar. Esse interesse em reunir a inovação no espaço social e tecnológico é o que me interessa mais sobre minha última área de pesquisa, a saúde móvel. Os desafios nesses projetos foram tão diversificados quanto os domínios de problema, mas o que todos eles tinham em comum foi a necessidade de manter os olhos abertos quanto a se tínhamos a definição correta do problema enquanto íamos e vínhamos entre projeto e desenvolvimento, protótipo e piloto. Nenhum desses foram problemas que poderiam ser solucionados analiticamente, com simulação ou mesmo nas experiências construídas em laboratório. Tudo isso desafiou nossa capacidade de reter arquiteturas limpas na presença de problemas e contextos confusos, e exigiu uma extensa colaboração.

Que mudanças e inovações você prevê que aconteçam nas redes sem fio e na mobilidade no futuro?

Nunca coloquei muita fé na previsão do futuro, mas diria que poderemos ver o fim dos telefones comuns (isto é, os que não são programáveis e são usados apenas para voz e mensagens de texto) à medida que smartphones se tornam mais e mais poderosos e são o ponto principal de acesso à Internet para muitos. Também acho que veremos a proliferação continuada de SIMs embutidas pelas quais todos os tipos de dispositivos têm a capacidade de se comunicar por meio da rede celular com baixas taxas de dados.

Que futuro você vê para as redes e a Internet?

Os esforços em dados nomeados e redes definidas por software surgirão para criar uma infraestrutura mais controlável, expansível e rica, representando de modo mais geral a mudança do papel da arquitetura para mais alto na pilha. Nos primórdios da Internet, a arquitetura ia até a camada 4, com as aplicações sendo mais monolíticas, no topo. Agora, dados e análise dominam o transporte.

Que pessoas a inspiraram profissionalmente?

Três pessoas me vêm à mente. Primeiro, Dave Clark, o tempero secreto e herói desconhecido da comunidade da Internet. Tive a sorte de estar por perto nos primeiros dias para vê-lo atuar como o "princípio organizador" do IAB e da governança na Internet; o sacerdote do consenso primitivo e do código em execução. Segundo, Scott Shenker, por seu brilhantismo intelectual, integridade e persistência. Procuro obter (mas raramente consigo) sua clareza na definição de problemas e soluções. Ele sempre é a primeira pessoa a quem envio e-mail pedindo conselho sobre coisas grandes e pequenas. Terceiro, minha irmã Judy Estrin, que teve a criatividade e coragem de passar toda a sua carreira levando ideias e conceitos ao mercado. Sem as Judys do mundo, as tecnologias da Internet nunca teriam transformado nossas vidas.

Quais são suas recomendações para alunos que desejam seguir carreira em ciência da computação e redes?

Primeiro, crie um alicerce forte em seu trabalho acadêmico, equilibrado com toda e qualquer experiência de trabalho do mundo real que possa conseguir. Ao procurar um ambiente de trabalho, busque oportunidades nas áreas de problema com que você realmente se importa e com pessoas com quem possa aprender.

C A P Í T U L O

1 2 3 4 5 6 7 8 9

REDES MULTIMÍDIA

Pessoas em todas as partes do mundo estão usando a Internet para assistir vídeos e shows de televisão por demanda. Empresas de distribuição de filmes e televisão, como Netflix e Hulu na América do Norte e Youku e Kankan na China praticamente se tornaram nomes corriqueiros. Mas as pessoas não estão apenas assistindo vídeos pela Internet, estão usando sites como YouTube para fazer *upload* e distribuir seu conteúdo gerado pelo próprio usuário, tornando-se produtores de vídeo e também consumidores. Além do mais, aplicações de rede como Skype, Google Talk e QQ (tremendamente populares na China) permitem que os usuários não apenas façam "ligações telefônicas" pela Internet, mas também as melhorem com vídeo e conferência entre várias pessoas. De fato, podemos com segurança prever que, ao final desta década, quase toda a distribuição de vídeo e interações de voz será feita de ponta a ponta pela Internet, muitas vezes para dispositivos sem fio conectados à Internet por meio de redes de acesso 4G e Wi-Fi.

Começaremos este capítulo com uma taxonomia das aplicações multimídia na Seção 7.1. Veremos que uma aplicação de multimídia pode ser classificada como *fluxo de áudio/vídeo armazenado, voz interativa/voz-sobre-IP* e *fluxo áudio/vídeo ao vivo*. Veremos que cada uma dessas classes de aplicações tem seus próprios requisitos de serviço exclusivos, que diferem significativamente daqueles das aplicações tradicionais, como e-mail, navegação Web e login remoto. Na Seção 7.2 examinaremos o fluxo de vídeo com mais atenção. Exploraremos muitos dos princípios básicos por trás do fluxo de vídeo, incluindo buffer do cliente, pré-busca e adaptação da qualidade do vídeo à largura de banda disponível. Também investigaremos as redes de distribuição de conteúdo (CDNs), que são bastante usadas hoje pelos principais sistemas de fluxo de vídeo. Depois examinamos os sistemas YouTube, Netflix e Kankan como estudos de caso para o fluxo de vídeo. Na Seção 7.3, analisaremos a voz e o vídeo coloquiais, que, diferentemente das aplicações elásticas, são altamente sensíveis ao atraso de fim a fim, mas podem tolerar uma perda de dados ocasional. Aqui, examinaremos como técnicas do tipo reprodução adaptativa, correção de erro de repasse e ocultação de erro podem mitigar perda de pacotes e atrasos induzidos pela rede. Também examinaremos o Skype como um estudo de caso. Na Seção 7.4, estudaremos RTP e SIP, dois protocolos populares para aplicações de voz e vídeo interativas em tempo real. Na Seção 7.5, investigaremos os mecanismos dentro da rede que podem ser usados para diferenciar uma classificação de tráfego (por exemplo, aplicações que toleram atrasos, como a voz interativa) de outras (por exemplo, aplicações elásticas, como a navegação de páginas Web), e fornecer um serviço diferenciado entre várias classificações de tráfego.

7.1 APLICAÇÕES DE REDE MULTIMÍDIA

Definimos uma aplicação de rede multimídia como qualquer aplicação de rede que empregue áudio ou vídeo. Nesta seção, oferecemos uma taxonomia das aplicações multimídia. Veremos que cada classe de aplicações tem seu próprio conjunto exclusivo de requisitos de serviço e questões de projeto. Porém, antes de nos aprofundarmos em uma discussão sobre aplicações multimídia da Internet, é útil considerar as características intrínsecas das próprias mídias de áudio e vídeo.

7.1.1 Propriedades de vídeo

Talvez a característica mais destacada do vídeo seja sua **alta taxa de bits**. O vídeo distribuído pela Internet costuma variar de 100 kbits/s para videoconferências de baixa qualidade até mais de 3 Mbits/s para os filmes de fluxo de vídeo com alta definição. Para ter uma ideia de como as demandas de largura de banda de vídeo são comparadas com aquelas de outras aplicações da Internet, vamos considerar de modo breve três usuários, cada um usando uma aplicação diferente na Internet. Nosso primeiro usuário, Frank, está vendo rapidamente as fotos postadas nas páginas do Facebook de seus amigos. Suponhamos que Frank esteja vendo uma nova foto a cada 10 segundos, e que as fotos tenham um tamanho médio de 200 Kbytes. (Como sempre, nesta discussão vamos simplificar e supor que 1 Kbyte = 8.000 bits.) Nosso segundo usuário, Marta, está baixando música da Internet ("a nuvem") para o seu smartphone. Vamos imaginar que Marta esteja escutando muitas canções em MP3, uma após a outra, cada uma codificada a uma taxa de 128 kbits/s. O terceiro usuário, Vítor, está assistindo vídeo que foi codificado a 2 Mbits/s. Por fim, vamos supor que o tamanho da sessão para todos os três usuários seja 4.000 segundos (cerca de 67 minutos). A Tabela 7.1 compara as taxas de bits e o total de bytes transferidos para esses três usuários. Vemos que o vídeo de fluxo contínuo consome, de longe, a maior largura de banda, tendo uma taxa de bits mais de dez vezes maior que a das aplicações de Facebook e fluxo de música. Portanto, ao projetar aplicações de vídeo em rede, a primeira coisa que precisamos ter em mente são os altos requisitos de taxa de bits do vídeo. Dada a popularidade do vídeo e sua alta taxa de bits, talvez não seja surpresa que a Cisco preveja [Cisco, 2011] que o vídeo de fluxo contínuo e armazenado será mais ou menos 90% do tráfego da Internet para o consumidor global em 2015.

Outra característica importante do vídeo é que ele pode ser compactado, compensando assim a qualidade com a taxa de bits. Um vídeo é uma sequência de imagens, em geral exibidas a uma velocidade constante, por exemplo, 24 ou 30 imagens por segundo. Uma imagem não compactada, codificada digitalmente, consiste em uma matriz de pixels, com cada pixel codificado em uma série de bits para representar luminosidade e cor. Existem dois tipos de redundância no vídeo, e ambos podem ser explorados pela **compactação de vídeo**. A *redundância espacial* é a que ocorre dentro de determinada imagem. De modo intuitivo, uma imagem que consiste principalmente em espaço em branco tem alto grau de redundância e pode ser compactada de maneira eficiente sem sacrificar a qualidade da imagem significativamente. A *redundância temporal* reflete a repetição de uma imagem para a seguinte. Por exemplo, se uma imagem e a seguinte forem idênticas, não há razão para codificar de novo a imagem seguinte; em vez disso, é mais eficiente apenas indicar, durante a codificação, que ela é exatamente a mesma. Os algoritmos de compactação de hoje, prontos para uso, podem compactar um vídeo basicamente para qualquer taxa de bits desejada. É claro que, quanto mais alta a taxa de bits, melhor a qualidade da imagem e melhor a experiência de exibição geral do usuário.

TABELA 7.1 COMPARAÇÃO DOS REQUISITOS DE TAXA DE BITS DE TRÊS APLICAÇÕES NA INTERNET

	Taxa de bits	Bytes transferidos em 67 min
Facebook de Frank	160 kbits/s	80 Mbytes
Música de Marta	128 kbits/s	64 Mbytes
Vídeo de Vítor	2 Mbits/s	1 Gbyte

Também podemos usar a compactação para criar **múltiplas versões** do mesmo vídeo, cada uma em um nível de qualidade diferente. Por exemplo, podemos usar a compactação para criar, digamos, três versões do mesmo vídeo, nas taxas de 300 kbits/s, 1 Mbit/s e 3 Mbits/s. Os usuários podem decidir qual versão eles querem ver como uma função da largura de banda disponível. Os usuários com conexões de alta velocidade com a Internet escolheriam a de 3 Mbits/s; os que assistem ao vídeo por 3G com um smartphone poderiam escolher a versão de 300 kbits/s. De modo semelhante, o vídeo em uma aplicação de videoconferência pode ser compactado "no ato" para oferecer a melhor qualidade dada a largura de banda de fim a fim disponível entre usuários conversando.

7.1.2 Propriedades de áudio

O áudio digital (incluindo a fala e a música digitalizadas) tem requisitos de largura de banda muito menores do que o vídeo. O áudio digital, porém, tem suas propriedades exclusivas, que devem ser consideradas quando se projetam aplicações de redes multimídia. Para entender essas propriedades, vamos primeiro considerar como o áudio analógico (que os humanos e os instrumentos musicais geram) é convertido para um sinal digital:

- O sinal analógico de áudio é primeiro amostrado a alguma taxa fixa; por exemplo, 8 mil amostras por segundo. O valor de cada amostra é um número real qualquer.
- Cada amostra é então arredondada para um valor entre um número finito de valores. Essa operação é denominada **quantização**. O número de valores finitos — denominados valores de quantização — em geral é uma potência de 2, por exemplo, 256 valores de quantização.
- Cada um dos valores de quantização é representado por um número fixo de bits. Por exemplo, se houver 256 valores de quantização, então cada valor — e, portanto, cada amostra de áudio — será representado por 1 byte. As representações por bits de todas as amostras são, então, concatenadas em conjunto para formar a representação digitalizada do sinal. Como exemplo, se um sinal de áudio for amostrado a uma taxa de 8 mil amostras por segundo e cada amostra for quantizada e representada por 8 bits, então o sinal digital resultante terá uma taxa de 64 mil bits por segundo. Esse sinal digital pode então ser reconvertido — isto é, decodificado — em um sinal analógico. Contudo, o sinal analógico decodificado é apenas uma aproximação do sinal de áudio original, e a qualidade do som pode ser nitidamente degradada (por exemplo, sons de alta frequência podem estar faltando no sinal decodificado). Aumentando a taxa de amostragem e o número de valores de quantização, o sinal decodificado pode se aproximar melhor do sinal analógico original. Assim, há uma clara permuta (como no vídeo) entre a qualidade do sinal decodificado e os requisitos de armazenamento e taxa de bits do sinal digital.

A técnica básica de codificação que acabamos de descrever é denominada **modulação por codificação de pulso** (*pulse code modulation* — PCM). A codificação de voz frequentemente usa PCM, com uma taxa de amostragem de 8 mil amostras por segundo e 8 bits por amostra, o que resulta uma taxa de 64 kbits/s. O disco compacto de áudio (CD) também usa PCM, com taxa de amostragem de 44.100 amostras por segundo e 16 bits por amostra; isso dá uma taxa de 705,6 kbits/s para mono e 1,411 Mbits/s para estéreo.

Entretanto, a voz e a música codificadas com PCM raramente são usadas na Internet. Em vez disso, como no vídeo, são usadas técnicas de compactação para reduzir as taxas de bits do fluxo. A voz humana pode ser compactada para menos de 10 kbits/s e ainda ser inteligível. Uma técnica de compactação popular para a música estéreo com qualidade quase de CD é **MPEG 1 layer 3**, mais conhecida como **MP3**. Codificadores MP3 podem compactar para muitas taxas diferentes; 128 kbits/s é a taxa de codificação mais comum, produzindo muito pouca degradação de som. Um padrão relacionado é o **Advanced Audio Coding (AAC)**, que foi popularizado pela Apple. Assim como o vídeo, diversas versões de um fluxo de áudio pré-gravado podem ser criadas, cada uma em uma taxa de bits diferente.

Embora as taxas de bit de áudio sejam em geral muito menores do que as de vídeo, os usuários costumam ser muito mais sensíveis a pequenas falha de áudio do que de vídeo. Considere, por exemplo, uma conferência de vídeo ocorrendo pela Internet. Se, de vez em quando, o sinal de vídeo for perdido por alguns segundos, a conferência de vídeo provavelmente poderá prosseguir sem muita frustração do usuário. Porém, se o sinal de áudio for perdido com frequência, os usuários poderão ter que terminar a sessão.

7.1.3 Tipos de aplicações de redes multimídia

A Internet pode comportar uma grande variedade de aplicações de multimídia úteis e divertidas. Nesta subseção, classificaremos as aplicações de multimídia em três categorias abrangentes: (i) *áudio e vídeo de fluxo contínuo armazenados*, (ii) *voz e vídeo-sobre-IP interativos* e (iii) *áudio e vídeo de fluxo contínuo ao vivo*. Como veremos em breve, cada uma dessas categorias de aplicação tem seu próprio conjunto de requisitos de serviço e questões de projeto.

Áudio e vídeo de fluxo contínuo armazenados

Para concretizar nossa discussão, focalizamos aqui o vídeo de fluxo contínuo armazenado, que em geral combina componentes de vídeo e áudio. O áudio de fluxo contínuo armazenado (como a música em fluxo contínuo) é muito semelhante ao vídeo de fluxo contínuo armazenado, embora as taxas de bits costumem ser muito menores.

Nesta classe de aplicações, o meio subjacente é o vídeo pré-gravado, como um filme, um show de TV, um evento esportivo pré-gravado ou um vídeo pré-gravado gerado pelo usuário (como aqueles vistos no YouTube). Esses vídeos são colocados em servidores e os usuários enviam solicitações aos servidores para verem os vídeos *por demanda*. Muitas empresas de Internet oferecem hoje vídeo de fluxo contínuo, incluindo YouTube (Google), Netflix e Hulu. Por algumas estimativas, o vídeo de fluxo contínuo armazenado contribui com mais de 50% do tráfego descendente nas redes de acesso à Internet atualmente [Cisco, 2011]. O vídeo de fluxo contínuo armazenado tem três características distintas importantes:

- *Fluxo contínuo*. Em uma aplicação de vídeo de fluxo contínuo armazenado, normalmente o cliente inicia a reprodução alguns segundos após começar a receber o vídeo do servidor. Isso significa que o cliente reproduz de uma parte do vídeo ao mesmo tempo em que recebe do servidor partes do arquivo que estão mais à frente. Essa técnica, conhecida como **fluxo contínuo** (*streaming*), evita ter de descarregar o arquivo inteiro (e incorrer em atraso potencialmente longo) antes de começar a reproduzi-lo.
- *Interatividade*. Como a mídia é pré-gravada, o usuário pode interromper, reposicionar para a frente, reposicionar para trás, avançar rapidamente, e assim por diante, pelo conteúdo do vídeo. O tempo desde quando o usuário faz essa solicitação até que a ação se manifeste no cliente deverá ser menor que alguns segundos para que haja reação aceitável.
- *Reprodução contínua*. Assim que se inicia a reprodução do vídeo, ela deve prosseguir de acordo com a temporização original da gravação. Portanto, os dados devem ser recebidos do servidor a tempo de ser reproduzidos no cliente; caso contrário, os usuários podem receber quadros de vídeo congelando (quando o cliente espera pelos quadros atrasados) ou saltando (quando o cliente pula os quadros atrasados).

De longe, a medida de desempenho mais importante para o vídeo de fluxo contínuo é a vazão média. Para oferecer reprodução contínua, a rede precisa oferecer uma vazão média à aplicação de fluxo contínuo que seja pelo menos tão grande quanto a taxa de bits do próprio vídeo. Conforme veremos na Seção 7.2, com o uso de buffers e pré-busca, é possível oferecer reprodução contínua mesmo quando a vazão flutua, desde que a vazão média (medida durante 5 a 10 segundos) permaneça acima da taxa do vídeo [Wang, 2008].

Para muitas aplicações de vídeo de fluxo contínuo, o vídeo pré-gravado é armazenado e enviado por CDN, em vez de um data center único. Há também muitas aplicações de vídeo de fluxo contínuo P2P, para as quais o vídeo é armazenado nos hospedeiros (pares) dos usuários, com diferentes pedaços do vídeo chegando de diversos pares, que podem estar espalhados por todo o globo. Dada a proeminência do vídeo de fluxo contínuo na Internet, exploraremos esse tema com mais detalhes na Seção 7.2, prestando atenção particular ao buffer do cliente, pré-busca, adaptação da qualidade à disponibilidade de largura de banda e distribuição CDN.

Voz e vídeo sobre IP interativos

A voz interativa em tempo real pela Internet é chamada de **telefonia da Internet**, visto que, do ponto de vista do usuário, é semelhante ao serviço telefônico tradicional, por comutação de circuitos. Isso em geral é cha-

mado de **Voz-sobre-IP (VoIP —** *Voice-Over-IP*). O vídeo interativo é semelhante, exceto que inclui o vídeo dos participantes, bem como suas vozes. A maioria dos sistemas interativos por voz e vídeo permite que os usuários criem conferências com três ou mais participantes. Voz e vídeo interativos são muito usados na Internet hoje, com as empresas Skype, QQ e Google Talk atraindo centenas de milhões de usuários diários.

Em nossa discussão sobre requisitos de serviço de aplicação no Capítulo 2 (Figura 2.4), identificamos algumas linhas pelas quais esses requisitos de aplicação podem ser classificados. Duas delas— considerações quanto à temporização e à tolerância à perda de dados — são de particular importância para aplicações interativas de voz e vídeo. Considerações de temporização são importantes porque muitas aplicações interativas de voz e vídeo são altamente **sensíveis a atraso**. Para uma interação com dois ou mais interlocutores, o atraso desde quando um usuário fala ou se move até que a ação seja manifestada na outra ponta deverá ser menor do que algumas centenas de milissegundos. Para a voz, atrasos menores que 150 ms não são percebidos por um ouvinte humano, atrasos entre 150 e 400 ms podem ser aceitáveis, e atrasos que ultrapassam 400 ms podem resultar em conversas de voz frustrantes, ou totalmente ininteligíveis.

Por outro lado, aplicações de multimídia interativas são **tolerantes à perda** — perdas ocasionais causam apenas pequenas perturbações na recepção de áudio e vídeo, e podem ser parcial ou totalmente encobertas. Essas características de sensibilidade a atraso e de tolerância à perda são claramente diferentes daquelas das aplicações elásticas, como a navegação Web, correio eletrônico, redes sociais e login remoto. Para aplicações elásticas, atrasos longos são incômodos, mas não particularmente prejudiciais; porém, a completude e a integridade dos dados transferidos são de suma importância. Exploraremos a voz e o vídeo interativos com mais detalhes na Seção 7.3, prestando especial atenção ao modo como a reprodução adaptativa, a correção de erro de repasse e a ocultação de erros podem aliviar a perda de pacotes e o atraso induzidos pela rede.

Áudio e vídeo de fluxo contínuo ao vivo

Esta terceira classe de aplicação é semelhante à transmissão tradicional de rádio e televisão, exceto que é realizada pela Internet. Essas aplicações permitem que um usuário receba uma transmissão de rádio ou televisão *ao vivo* de qualquer parte do mundo (por exemplo, um evento esportivo ao vivo ou um evento de notícias contínuas). Milhares de estações de rádio e televisão do mundo inteiro estão transmitindo conteúdo pela Internet.

Aplicações ao vivo, do tipo de difusão, normalmente possuem muitos usuários que recebem o mesmo programa de áudio/vídeo ao mesmo tempo. Embora a distribuição de áudio/vídeo ao vivo para muitos receptores possa ser realizada com eficiência utilizando as técnicas de transmissão IP para um grupo, conforme estudamos na Seção 4.7, a distribuição para um grupo é mais comumente feita por meio de aplicações em camadas (usando redes P2P ou CDN), ou de múltiplos fluxos individuais separados. Como acontece com a multimídia armazenada de fluxo contínuo, a rede precisa oferecer a cada fluxo de multimídia ao vivo uma vazão média que seja maior que a taxa de consumo de vídeo. Como o evento é ao vivo, o atraso também pode ser um problema, embora as limitações de temporização sejam menos severas do que para aplicações interativas de voz. Atrasos de até dez segundos ou mais desde o instante em que o usuário requisita a entrega/reprodução de uma transmissão ao vivo até o início da reprodução podem ser tolerados. Não estudaremos a mídia de fluxo contínuo ao vivo neste livro, pois muitas das técnicas usadas para a mídia de fluxo contínuo ao vivo — atraso de buffer inicial, uso de largura de banda adaptativa e distribuição CDN — são semelhantes àquelas para a mídia de fluxo contínuo armazenado.

7.2 VÍDEO DE FLUXO CONTÍNUO ARMAZENADO

Para aplicações de vídeo de fluxo contínuo, os vídeos pré-gravados são armazenados em servidores aos quais os usuários enviam solicitações para verem os vídeos por demanda. O usuário pode assistir o vídeo do início ao fim, sem interrupção, pode parar de assistir o vídeo antes que ele termine ou interagir com o vídeo

interrompendo ou reposicionando para uma cena futura ou passada. Os sistemas de vídeo de fluxo contínuo podem ser classificados em três categorias: **UDP de fluxo contínuo**, **HTTP de fluxo contínuo** e **HTTP de fluxo contínuo adaptativo**. Embora todos os três tipos de sistemas sejam usados na prática, a maioria dos sistemas de hoje emprega o HTTP de fluxo contínuo e o HTTP de fluxo contínuo adaptativo.

Uma característica comum de todas as três formas de vídeo de fluxo contínuo é o uso extenso de buffer de aplicação no lado do cliente para aliviar os efeitos de variar os atrasos de fim a fim e variar as quantidades de largura de banda disponível entre servidor e cliente. Para o vídeo de fluxo contínuo (tanto armazenado quanto ao vivo), os usuários podem tolerar um pequeno atraso inicial de alguns segundos entre o momento em que o cliente solicita um vídeo e quando a reprodução inicia no cliente. Por conseguinte, quando o vídeo começa a chegar no cliente, um cliente não precisa iniciar a reprodução imediatamente, mas pode acumular alguma reserva de vídeo em um buffer de aplicação. Quando o cliente tiver acumulado uma reserva de alguns segundos de vídeo mantido em buffer, mas não reproduzido, poderá iniciar a reprodução do vídeo. Existem duas vantagens importantes fornecidas por tal **buffer de cliente**. Primeiro, o buffer no lado do cliente pode absorver variações no atraso entre servidor e cliente. Se um trecho de vídeo se atrasar, desde que ele chegue antes que a reserva de vídeo recebido mas não reproduzido se esgote, esse atraso longo não será observado. Segundo, se a largura de banda do servidor ao cliente cair depressa para menos do que a taxa de consumo de vídeo, um usuário pode continuar a aproveitar a reprodução contínua, de novo desde que o buffer da aplicação cliente não seja completamente esgotado.

A Figura 7.1 ilustra o buffer no lado do cliente. Neste exemplo simples, suponha que o vídeo seja codificado a uma taxa de bits fixa, e assim cada bloco de vídeo contenha quadros de vídeo que devem ser reproduzidos, mas sobre a mesma quantidade de tempo, Δ. Um servidor transmite o primeiro bloco de vídeo em t_0, o segundo bloco em $t_0 + \Delta$, o terceiro bloco em $t_0 + 2\Delta$, e assim por diante. Quando o cliente inicia a reprodução, cada bloco deve ser reproduzido Δ unidades de tempo após o bloco anterior, a fim de reproduzir a temporização do vídeo gravado original. Por causa dos atrasos variáveis da rede de fim a fim, diferentes blocos de vídeo experimentam diferentes atrasos. O primeiro bloco de vídeo chega ao cliente em t_1 e o segundo bloco chega em t_2. O atraso da rede para o *i*-ésimo bloco é a distância horizontal entre o momento em que o bloco foi transmitido pelo servidor e o momento em que ele é recebido no cliente; observe que o atraso da rede varia de um bloco de vídeo para outro. Neste exemplo, se o cliente tivesse de começar a reprodução assim que o primeiro bloco chegasse em t_1, então o segundo bloco não teria chegado a tempo para ser reproduzido em $t_1 + \Delta$. Nesse caso, a reprodução teria que ser adiada (esperando até que o bloco 1 chegasse) ou o bloco 1 poderia ser pulado — ambos resultando em prejuízos indesejáveis na reprodução. Em vez disso, se o cliente tivesse de adiar o início da reprodução até t_3, quando todos os blocos de 1 a 6 tivessem chegado, a reprodução periódica poderia prosseguir com *todos* os blocos tendo sido recebidos antes do seu tempo de reprodução.

FIGURA 7.1 ATRASO DE REPRODUÇÃO DO CLIENTE NO VÍDEO DE FLUXO CONTÍNUO

7.2.1 UDP de fluxo contínuo

Até aqui, só falamos rapidamente sobre o UDP de fluxo contínuo, levando o leitor a procurar discussões mais profundas dos protocolos por trás desses sistemas, quando for o caso. Com o UDP de fluxo contínuo, o servidor transmite vídeo a uma taxa que corresponde à taxa de consumo de vídeo do cliente, com uma temporização dos trechos de vídeo sobre UDP a uma taxa constante. Por exemplo, se a taxa de consumo for 2 Mbits/s e cada pacote UDP transportar 8.000 bits de vídeo, o servidor transmitiria um pacote UDP em seu *socket* a cada (8.000 bits)/(2 Mbits/s) = 4 ms. Conforme aprendemos no Capítulo 3, como o UDP não emprega um mecanismo de controle de congestionamento, o servidor pode empurrar pacotes na rede na taxa de consumo do vídeo sem as restrições de controle de taxa do TCP. O UDP de fluxo contínuo normalmente usa um pequeno buffer no lado do cliente, grande o suficiente para manter menos de um segundo de vídeo.

Antes de passar os trechos de vídeo ao UDP, o servidor encapsulará os trechos de vídeo dentro de pacotes de transporte projetados especialmente para transportar áudio e vídeo, usando o Real-Time Transport Protocol (RTP) [RFC 3550] ou um esquema semelhante (possivelmente, patenteado). Deixaremos nossa explicação sobre RTP para a Seção 7.3, na qual discutimos o RTP no contexto dos sistemas interativos de voz e vídeo.

Outra propriedade distinta do UDP de fluxo contínuo é que, além do fluxo de vídeo do servidor ao cliente, cliente e servidor também mantêm, em paralelo, uma conexão de controle separada sobre a qual o cliente envia comandos referentes a mudanças de estado de sessão (como pausar, retomar, reposicionar e assim por diante). Essa conexão de controle é, de várias maneiras, semelhante à conexão de controle FTP que estudamos no Capítulo 2. O Real-Time Streaming Protocol (RTSP) [RFC 2326], explicado com detalhes no site deste livro, é um protocolo aberto popular para esse tipo de conexão de controle.

Embora o UDP de fluxo contínuo tenha sido empregado em muitos sistemas de fonte aberta e produtos patenteados, ele possui três desvantagens significativas. Primeiro, pela quantidade imprevisível e variável de largura de banda disponível entre servidor e cliente, o UDP de fluxo contínuo com taxa constante pode deixar de oferecer reprodução contínua. Por exemplo, considere o cenário onde a taxa de consumo de vídeo é de 1 Mbit/s e a largura de banda disponível do servidor ao cliente normalmente é maior que 1 Mbit/s, mas a cada minuto ou mais a largura de banda disponível cai para menos de 1 Mbit/s por vários segundos. Nesse cenário, um sistema de UDP de fluxo contínuo que transmite vídeo a uma taxa constante de 1 Mbit/s por RTP/UDP provavelmente não agradaria ao usuário, com quadros congelando ou pulando logo depois que a largura de banda disponível caísse para menos de 1 Mbit/s. A segunda desvantagem é que ele exige um servidor de controle de mídia, como um servidor RTSP, para processar solicitações de interatividade cliente-servidor e acompanhar o estado do cliente (por exemplo, o ponto de reprodução do cliente no vídeo, esteja o vídeo sendo interrompido ou reproduzido, e assim por diante) para *cada* sessão do cliente em andamento. Isso aumenta o custo geral e a complexidade da implantação de um sistema de vídeo por demanda em grande escala. A terceira desvantagem é que muitos firewalls são configurados para bloquear o tráfego UDP, impedindo que os usuários por trás desses firewalls recebam o vídeo UDP.

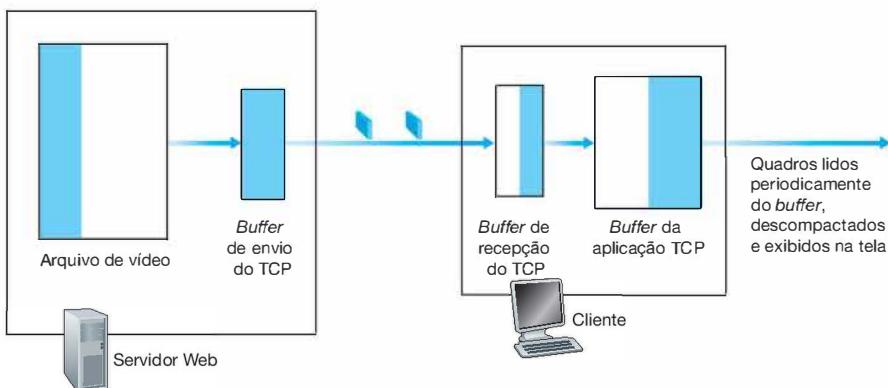
7.2.2 HTTP de fluxo contínuo

No HTTP de fluxo contínuo, o vídeo é apenas armazenado em um servidor HTTP como um arquivo comum com uma URL específica. Quando um usuário quer assistir a este vídeo, ele estabelece uma conexão TCP com o servidor e realiza um comando HTTP GET para aquele URL. O servidor envia então o arquivo de vídeo, dentro de uma mensagem de resposta HTTP, o mais rápido possível, isto é, tão depressa quanto o controle de congestionamento TCP e o controle de fluxo permitirem. No cliente, os bytes são armazenados em um buffer de aplicação cliente. Uma vez que o número de bytes neste buffer exceder um limite predeterminado, a aplicação cliente inicia uma reprodução — mais especificamente, ela captura quadros do vídeo do buffer da aplicação cliente, descompacta os quadros e os apresenta na tela do usuário.

Aprendemos no Capítulo 3 que, quando transferimos um arquivo usando TCP, a taxa de transmissão do servidor para o cliente pode variar significativamente por causa do mecanismo de controle de congestionamen-

to do TCP. Mais especificamente, não é raro para a taxa de transmissão variar no formato "dente de serra" (por exemplo, Figura 3.53) associado com o controle de congestionamento do TCP. Além disso, os pacotes podem também sofrer atraso significativo, pelo mecanismo de retransmissão do TCP. Por causa destas características do TCP, o consenso geral era que fluxo contínuo de vídeo nunca funcionaria bem sobre TCP. Ao longo do tempo, porém, projetistas de sistemas de fluxo contínuo de vídeo aprenderam que o controle de congestionamento do TCP e os mecanismos de transferência de dados confiáveis não necessariamente impedem a emissão quando o buffer do cliente e pré-busca (discutido na próxima seção) são usados.

O uso do HTTP sobre TCP também permite ao vídeo atravessar firewalls e NATs mais facilmente (os quais são em geral configurados para bloquear a maior parte do tráfego UDP, mas permitir o tráfego HTTP). Vídeos de fluxo contínuo sobre HTTP também deixam clara a necessidade de um servidor de controle de mídia, tal como um servidor RTSP, de modo a reduzir o custo de um desenvolvimento em larga escala pela Internet. Por todas essas vantagens, a maior parte dos aplicativos de fluxo contínuo de vídeo — incluindo YouTube e Netflix — usam HTTP de fluxo contínuo (sobre TCP) como seu protocolo de fluxo contínuo subjacente.

Pré-busca de vídeo

Como acabamos de aprender, o uso de buffers no lado do cliente pode ser usado para mitigar os efeitos de vários atrasos fim a fim e variadas larguras de banda disponíveis. Em nosso último exemplo na Figura 7.1, o servidor transmite vídeo a uma taxa na qual deverá ser exibido. Porém, para vídeos de fluxo contínuo *armazenado*, o cliente pode tentar baixar o vídeo em uma taxa *maior* que a de consumo, e assim fazer uma **pré-busca** dos quadros desse vídeo que serão exibidos no futuro. Esse vídeo previamente baixado é armazenado no buffer de aplicação do cliente. A pré-busca ocorre naturalmente com o TCP de fluxo contínuo, uma vez que o mecanismo que impede o congestionamento tentará usar toda a largura de banda disponível entre o cliente e o servidor.

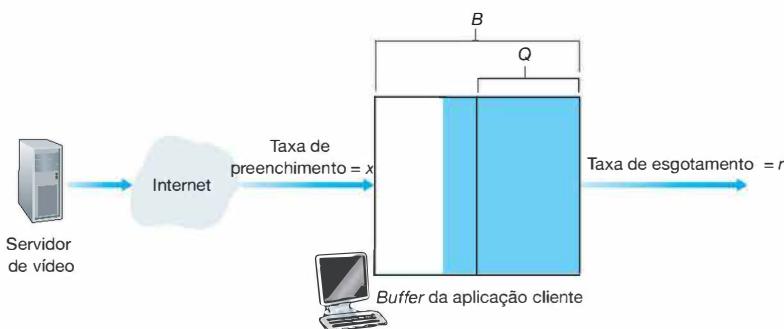
Para entendermos melhor a pré-busca, vamos observar um exemplo simples. Suponha que a taxa de consumo do vídeo seja 1 Mbit/s mas a rede é capaz de transmitir o vídeo do servidor para o cliente em uma taxa constante de 1,5 Mbits/s. Então o cliente será capaz, não só de exibir o vídeo com um atraso de reprodução muito pequeno, como também conseguirá aumentar a quantidade de dados do vídeo armazenado no buffer a uma taxa de 500 Kbits a cada segundo. Desta maneira, se no futuro o cliente receber dados a uma taxa menor que 1 Mbit/s, por um pequeno período de tempo, ao cliente será possível continuar a exibir sem interrupções graças à reserva em seu buffer. Wang [2008] nos mostra que, quando a taxa de transferência disponível do TCP for aproximadamente o dobro da taxa de bits da mídia, o fluxo contínuo sobre TCP resultará em uma mínima inanição e baixos atrasos no uso do buffer.

Buffer de aplicação do cliente e buffers TCP

A Figura 7.2 ilustra a interação entre o cliente e o servidor para HTTP de fluxo contínuo. No lado do servidor, a parte do arquivo de vídeo em branco já foi enviada dentro do *socket* do servidor, enquanto a parte mais escura é o que falta ser enviado. Depois de "passar pela porta do *socket*", os bytes são colocados no buffer de envio do TCP antes de serem transmitidos na Internet, como descrito no Capítulo 3. Na Figura 7.2, como o buffer de envio do TCP é apresentado como cheio, o servidor momentaneamente é impedido de enviar mais bytes do arquivo de vídeo para o *socket*. No lado do cliente, a aplicação cliente (o tocador de mídia) lê os bytes que estão no buffer de recepção do TCP (através do *socket* do cliente) e coloca os bytes no buffer da aplicação cliente. Ao mesmo tempo, a aplicação cliente periodicamente retém quadros de vídeo do buffer da aplicação cliente, descompacta os quadros e os apresenta na tela do usuário. Note que, se o buffer da aplicação cliente for maior que o arquivo do vídeo, então o processo completo de movimentação dos bytes do armazenamento no servidor para o buffer da aplicação cliente é equivalente a baixar um arquivo comum usando HTTP — o cliente apenas obtém o vídeo do servidor tão rápido quanto o TCP permitir!

Considere agora o que acontece quando o usuário pausa o vídeo durante o processo de fluxo contínuo. No período de pausa, bits não são removidos do buffer da aplicação cliente, embora continuem a entrar no buffer, vindos

FIGURA 7.2 VÍDEO DE FLUXO CONTÍNUO ARMAZENADO COM HTTP/TCP

do servidor. Se o buffer da aplicação cliente for finito, ele pode eventualmente ficar cheio, o que causaria "pressão contrária" por todo o caminho de volta até o servidor. Mais especificamente, uma vez que o buffer da aplicação cliente fique cheio, bytes não poderão mais ser removidos do buffer de recepção do TCP no cliente, então ele também ficará cheio. Desde que o buffer de recepção do TCP no cliente se encha, bytes não poderão mais ser removidos do buffer de envio do TCP no cliente, então ele também ficará cheio. E uma vez que o buffer de envio do TCP no cliente fique cheio, o servidor não poderá mais enviar byte algum para o *socket*. Sendo assim, se o usuário pausar o vídeo, o servidor poderá ser forçado a parar de transmitir, e neste caso será bloqueado até que o usuário volte a exibir o vídeo.

De fato, mesmo durante uma reprodução regular (isto é, sem pausa), se o buffer da aplicação cliente ficar cheio, uma pressão contrária fará os buffers do TCP ficarem cheios, o que forçará o servidor a reduzir sua taxa de transmissão. Para determinar a taxa resultante, note que, quando a aplicação cliente remove f bits, ela cria espaço para f bits no buffer da aplicação cliente, que por sua vez permitirá ao servidor enviar f bits adicionais. Sendo assim, a taxa de envio do servidor não pode ser maior que a taxa de consumo do vídeo no cliente. Portanto, *um* buffer *da aplicação cliente cheio indiretamente impõe um limite sobre a taxa que o vídeo poderá ser enviado do servidor para o cliente quando estiver usando fluxo contínuo sobre HTTP.*

Análise do vídeo de fluxo contínuo

Alguns modelos simples oferecerão um melhor entendimento sobre o atraso inicial na reprodução e o congelamento devido ao esgotamento do buffer da aplicação. Como mostrado na Figura 7.3, B representa o tamanho (em bits) do buffer da aplicação cliente, e Q indica o número de bits que têm de ser armazenados no buffer antes que a aplicação cliente comece a exibir o vídeo. ($Q < B$, é claro.) A letra r representa a taxa de consumo do vídeo — a taxa na qual o cliente retira bits do buffer da aplicação cliente durante a reprodução. Então, por exemplo, se a taxa de quadros do vídeo é 30 quadros/segundo, e cada quadro (compactado) tem 100 mil bits, então r = 3 Mbits/s. Para ver a floresta por entre as árvores, iremos ignorar os buffers do TCP de recepção e de envio.

Vamos supor que o servidor envia bits a uma taxa constante x sempre que o buffer do cliente não estiver cheio. (Esta é uma simplificação grosseira, pois a taxa de envio do TCP varia por causa do controle de congestionamento; vamos examinar de forma mais realista as taxas como função do tempo $x(t)$ nos problemas do final deste capítulo.) Suponha que, no tempo $t = 0$, o buffer da aplicação está vazio e o vídeo começa a chegar no buffer da aplicação cliente. Agora perguntamos: em que tempo $t = t_p$ o vídeo começará a ser exibido? E durante a reprodução, em que tempo $t = t_f$ o buffer da aplicação cliente ficará cheio?

Primeiro, vamos determinar t_p, o tempo que leva para que Q bits tenham entrado no buffer da aplicação cliente e a reprodução comece. Lembre que os bits chegam ao buffer da aplicação cliente a uma taxa x e *nenhum*

FIGURA 7.3 ANÁLISE DO USO DE BUFFER NO LADO DO CLIENTE, PARA VÍDEO DE FLUXO CONTÍNUO

bit é removido antes de a reprodução começar. Então, a quantidade de tempo necessária para acumular Q bits (o atraso inicial do buffer) é $t_p = Q/x$.

Agora vamos determinar t_f, o momento no tempo quando o buffer da aplicação cliente fica cheio. Primeiro devemos observar que, se $x < r$ (ou seja, se a taxa de envio do servidor for menor que a taxa de consumo do vídeo), então o buffer do cliente nunca ficará cheio! De fato, iniciando no tempo t_p, o buffer estará esgotado em uma taxa r e estará preenchido em uma taxa $x < r$. Eventualmente o buffer do cliente se esvaziará completamente, no momento em que o vídeo congelar na tela enquanto o buffer do cliente espera outros t_p segundos para acumular Q bits de vídeo. Então, quando a taxa disponível na rede for menor que a taxa do vídeo, a reprodução irá alternar entre períodos de reprodução contínua e períodos de congelamento. Em um dever de casa, você será solicitado a determinar o tamanho de cada período de reprodução contínua e de congelamento, como função de Q, r e x. Agora vamos determinar t_f, quando $x > r$. Neste caso, começando no tempo t_p, o buffer aumenta de Q até B a uma taxa $x - r$ desde que os bits sejam retirados a uma taxa r mas cheguem a uma taxa x, como mostrado na Figura 7.3. Com as dicas apresentadas, será solicitado, em um problema de dever de casa, determinar t_f, o tempo que o buffer do cliente leva para ficar cheio. Observe que, *quando a taxa disponível na rede for maior que a taxa do vídeo, após o atraso inicial do* buffer, *o usuário irá desfrutar de uma reprodução contínua até que a reprodução do vídeo termine.*

Término antecipado e reposicionamento do vídeo

Sistemas de HTTP de fluxo contínuo fazem uso com frequência do **cabeçalho do intervalo de bytes HTTP** na mensagem da requisição do HTTP GET, o qual determina a faixa específica de bytes que o cliente no momento quer buscar do vídeo desejado. Isto é útil em particular quando o usuário quer reposicionar (ou seja, saltar) para um ponto adiante no vídeo. Quando o usuário reposiciona para uma nova posição, o cliente envia uma nova requisição HTTP, indicando no cabeçalho do intervalo de bytes a partir de qual byte no arquivo o servidor deve enviar dados. Quando o servidor recebe uma nova requisição HTTP, pode esquecer qualquer requisição anterior e, em vez disso, enviar bytes começando daquele indicado na requisição de intervalo de bytes.

Enquanto estamos tratando do assunto reposicionamento, mencionamos por alto o fato de que, quando um usuário reposiciona para um ponto adiante no vídeo ou termina o vídeo antecipadamente, alguns dados "pré-buscados mas ainda não visualizados", transmitidos pelo servidor, permanecerão sem ser assistidos — um desperdício de largura de banda da rede e de recursos do servidor. Por exemplo, suponha que o buffer do cliente esteja cheio com B bits em algum tempo t_0 no vídeo, e neste tempo o usuário reposiciona em algum instante $t > t_0 + B/r$, e então assiste ao vídeo até o fim a partir dali. Neste caso, todos os B bits no buffer não serão mais assistidos, assim como a largura de banda e os recursos do servidor que foram usados para transmitir esses B bits foram completamente desperdiçados. Existe um significativo desperdício de largura de banda na Internet por causa de términos antecipados de reprodução,

o que pode ser realmente custoso, em particular para conexões sem fio [Ihm, 2011]. Por esta razão, muitos sistemas de fluxo contínuo definem um tamanho apenas moderado para o buffer da aplicação cliente, ou limitarão a quantidade de vídeos pré-buscados usando o cabeçalho de intervalo de bytes nas requisições HTTP [Rao, 2011].

Reposicionamento e término antecipado podem ser comparados a cozinhar um grande bife, comer só uma parte dele e jogar o resto fora, logo, desperdiçando comida. Então, da próxima vez que seus pais o criticarem por desperdiçar alimento porque não comeu todo o seu jantar, você pode rapidamente responder dizendo que eles também estão dissipando largura de banda e recursos do servidor quando reposicionam enquanto assistem a filmes pela Internet! Mas, é claro, dois erros não fazem um acerto — tanto comida quanto largura de banda não devem ser desperdiçadas!

7.2.3 Fluxo contínuo adaptativo e DASH

Apesar de o HTTP de fluxo contínuo, como descrito nas subseções anteriores, estar sendo amplamente utilizado na prática (por exemplo, pelo YouTube desde o seu início), ele possui uma grande deficiência: todos os clientes recebem a mesma codificação do vídeo, apesar das grandes variações na quantidade de largura de banda disponível para o cliente, tanto através de diferentes clientes quanto ao longo do tempo para um mesmo cliente. Isto levou ao desenvolvimento de um novo tipo de fluxo contínuo baseado em HTTP, comumente referido como **Fluxo Contínuo Adaptativo Dinamicamente sobre HTTP (DASH)**. Pelo DASH, o vídeo é codificado em muitas versões diferentes, cada qual com uma taxa de bits e um diferente nível de qualidade. O cliente requisita dinamicamente, dessas diferentes versões, trechos de alguns segundos de segmentos do vídeo. Quando a quantidade de largura de banda disponível é alta, o cliente em geral seleciona trechos de uma versão que possui uma taxa alta; e quando a largura de banda disponível é baixa, ele naturalmente seleciona de uma versão cuja taxa é baixa. O cliente seleciona diferentes trechos um de cada vez com mensagens de requisição HTTP GET [Akhshabi, 2011].

Por um lado, DASH permite aos clientes com diferentes taxas de acesso à Internet fluir em um vídeo por diferentes taxas codificadas. Clientes com conexões 3G lentas podem receber uma versão com baixa taxa de bits (e baixa qualidade), e clientes com conexões por fibra ótica podem receber versões de alta qualidade. Por outro lado, o uso de DASH permite a um cliente se adaptar à largura de banda disponível, se a largura de banda fim a fim mudar durante a sessão. Esta funcionalidade é particularmente importante para usuários de dispositivos móveis, que com frequência experimentam flutuações na largura de banda disponível, conforme se movimentam de uma estação de base para outra. Por exemplo, a Comcast tem implantado um sistema de fluxo contínuo adaptativo no qual cada arquivo fonte de vídeo é codificado em 8 a 10 diferentes formatos MPEG-4, permitindo assim que o formato de vídeo de maior qualidade possível seja acessado pelo cliente, sendo a adaptação feita em resposta a mudanças na rede e nas condições dos dispositivos.

Com o DASH, cada versão do vídeo é armazenada em um servidor HTTP, cada um com uma diferente URL. O servidor HTTP também possui um **arquivo de manifesto**, que provê uma URL para cada versão junto com a sua taxa de bits. Primeiro, o cliente requisita o arquivo de manifesto e identifica as várias versões. O cliente, então, seleciona um trecho de cada vez, especificando uma URL e um intervalo de bytes em uma mensagem de requisição HTTP GET, para cada trecho. Enquanto estiver baixando trechos, o cliente também mede a largura de banda de recepção e executa um *algoritmo de determinação de taxa* para selecionar o próximo trecho a ser requisitado. Naturalmente, se o cliente possui muito vídeo em seu buffer e se a largura de banda de recepção que foi medida for alta, ele escolherá um trecho de uma versão com taxa alta associada. E, claro, se o cliente possui pouco vídeo em seu buffer e a sua largura de banda de recepção for baixa, escolherá um trecho de uma versão de taxa baixa. Portanto, DASH permite ao cliente alternar livremente em diferentes níveis de qualidade. Uma vez que uma súbita queda na taxa de bits, por selecionar outra versão, pode acarretar uma degradação na qualidade visual facilmente perceptível, a redução na taxa de bits pode ser realizada usando múltiplas versões intermediárias e assim possibilitar uma transição mais suave para uma taxa em que a taxa de consumo do cliente diminua abaixo de sua largura de banda disponível. Quando as condições da rede melhorarem, o cliente pode então depois escolher trechos de versões com maior taxa de bits.

Pelo monitoramento dinâmico da largura de banda disponível e do nível do buffer do cliente, e ajustando a taxa de transmissão pela troca de versão, o uso de DASH geralmente pode efetuar uma reprodução contínua no melhor nível de qualidade possível, sem o problema de congelamento ou salto de quadros. Além disso, uma vez que o cliente (ao invés do servidor) mantenha a inteligência para determinar qual trecho deve ser enviado em seguida, este esquema também aumenta a escalabilidade no lado do servidor. Outro benefício dessa abordagem é o cliente poder usar a requisição de faixa de bytes do HTTP para controlar com precisão a quantidade de vídeo obtido por pré-busca, o qual é armazenado localmente.

Concluímos nossa breve discussão sobre DASH mencionando que, para muitas execuções, o servidor não apenas armazena diversas versões do vídeo, como também armazena em separado muitas versões do áudio. Cada versão do áudio possui seu próprio nível de qualidade e taxa de bits, assim como sua própria URL. Nessas execuções, o cliente dinamicamente seleciona trechos tanto do vídeo como do áudio, e localmente sincroniza a reprodução do áudio e vídeo.

7.2.4 Redes de distribuição de conteúdo

Muitas companhias de vídeo na Internet têm distribuído sob demanda fluxos contínuos multi-Mbits/s para milhões de usuários diariamente. YouTube, por exemplo, com uma biblioteca de centenas de milhões de vídeos, distribui centenas de milhões de vídeos de fluxo contínuo para usuários ao redor do mundo inteiro, todos os dias [Ding, 2011]. Controlar o fluxo contínuo de todo esse tráfego para locais ao redor do mundo inteiro, enquanto provê reprodução contínua e grande interatividade constitui-se claramente uma tarefa desafiadora.

Para uma empresa de vídeos na Internet, talvez a mais franca abordagem para prover serviços de vídeo de fluxo contínuo seja construir um único e sólido datacenter, armazenar ali todos os vídeos e realizar o fluxo contínuo dos vídeos diretamente do datacenter para os clientes ao redor do mundo. No entanto, existem três grandes problemas com essa abordagem. Primeiro, se o cliente estiver muito longe do datacenter, os pacotes que vão do servidor para o cliente atravessarão muitos enlaces de comunicação, e também possivelmente passarão por muitos ISPs com alguns destes ISPs talvez localizados em diferentes continentes. Se um dos enlaces fornecer uma taxa de transferência menor que a de consumo do vídeo, a taxa de transferência fim a fim será também abaixo da de consumo, resultando para o usuário em atrasos incômodos por congelamento. (Lembre-se, do Capítulo 1, que uma taxa de transferência fim a fim de um fluxo contínuo é controlada pela taxa de transferência de seu enlace de gargalo.) A probabilidade de isso acontecer cresce na mesma proporção em que o número de enlaces no caminho fim a fim aumenta. Uma segunda desvantagem é que um vídeo popular será possivelmente enviado muitas vezes pelos mesmos enlaces de comunicação. Isto não apenas é um desperdício de largura de banda de rede, mas a própria companhia de vídeo pela Internet estará pagando pelo seu ISP (conectado com o datacenter) por enviar os mesmos bytes pela Internet muitas e muitas vezes. Um terceiro problema com essa solução é que um único datacenter representa um único ponto de falha — se o datacenter ou seus enlaces para a Internet caírem, ele não será capaz de distribuir *nenhum* vídeo de fluxo de dados.

Para enfrentar o desafio de distribuição de uma quantidade maciça de dados de vídeo para usuários espalhados ao redor do mundo, quase todas as maiores companhias de vídeo de fluxo contínuo fazem uso de **Redes de Distribuição de Conteúdo (CDNs)**. Uma CDN gerencia servidores em múltiplas localidades distribuídas geograficamente, armazena cópias dos vídeos (e outros tipos de conteúdos da rede, incluindo documentos, imagens e áudio) em seus servidores, e tenta direcionar cada requisição do usuário para uma localidade CDN que proporcionará a melhor experiência para o usuário. A CDN pode ser uma **CDN privada**, isto é, que pertence ao próprio provedor de conteúdo; por exemplo, a CDN da Google distribui vídeos do YouTube e outros tipos de conteúdo. A CDN também pode ser uma **CDN de terceiros**, que distribui conteúdo em nome de múltiplos provedores de conteúdo; a da Akamai, por exemplo, é uma CDN de terceiros que distribui conteúdos do Netflix e da Hulu, dentre outros. Uma visão geral das modernas CDNs, que é muito lida, pode ser vista em Leighton [2009].

As CDNs adotam em geral uma entre as duas filosofias de instalação de servidores [Huang, 2008]:

- ***Enter deep.*** Uma filosofia, que começou a ser praticada pela Akamai, é entrar profundamente dentro das redes de acesso dos Provedores de Serviço de Internet pela implantação de *clusters* de servidores no acesso de ISPs por todo o mundo. (Redes de acesso são descritas na Seção 1.3.) A Akamai usa esta abordagem com *clusters* de servidores em cerca de 1.700 locais. O objetivo é conseguir proximidade com os usuários finais, melhorando assim o atraso percebido pelo usuário e a taxa de transferência pela diminuição do número de enlaces e roteadores entre o usuário final e o agrupamento da CDN da qual ele recebe conteúdo. Por causa desse projeto altamente distribuído, a tarefa de manter e gerenciar os agrupamentos se torna desafiadora.
- ***Bring home.*** Uma segunda filosofia de projeto, adotada pela Limelight e por muitas outras companhias de CDN é *trazer para dentro de casa os ISPs*, construindo *clusters* enormes, mas em um número menor (por exemplo, dezenas) de lugares-chave e conectando-os em uma rede privada de alta velocidade. Em vez de entrar nos ISPs de acesso, estas CDNs normalmente colocam cada *cluster* em uma localidade que seja ao mesmo tempo próxima aos PoPs (veja a Seção 1.3) de muitos ISPs da camada 1, por exemplo, a algumas milhas tanto da AT&T quanto aos PoPs da Verizon em uma cidade maior. Comparado com a filosofia de projeto *enter deep*, o projeto *bring home* em geral resulta em menos desperdício com gerenciamento e manutenção, mas com um maior atraso e menores taxas de transferência para os usuários finais.

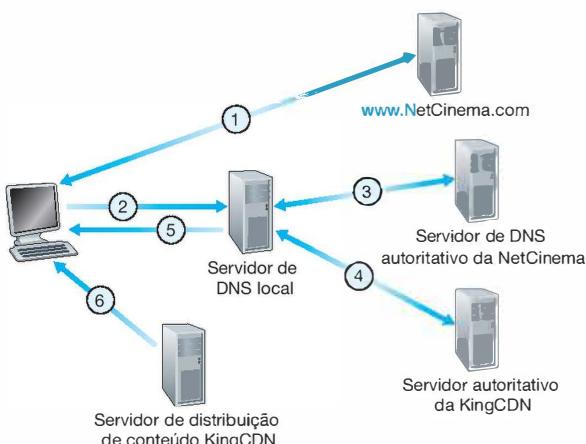
ESTUDO DE CASO

Infraestrutura de rede da Google

Para suportar sua vasta lista de serviços na nuvem — incluindo busca, gmail, calendário, vídeos do YouTube, mapas, documentos e redes sociais — a Google implementou uma rede privada extensiva de infraestrutura de CDN. A infraestrutura de CDN da Google possui três camadas de agrupamentos de servidores:

- Oito "mega datacenters", com seis localizados nos Estados Unidos e dois na Europa [Google Locations, 2012], com cada datacenter contendo algo na ordem de 100 mil servidores. Esses mega datacenters são responsáveis por servir conteúdo dinâmico (e muitas vezes personalizado), incluindo resultados de busca e mensagens de gmail.
- Cerca de 30 *clusters bring home* (veja discussão na Seção 7.2.4), com cada *cluster* consistindo algo na ordem de 100-500 servidores [Adhikari, 2011a]. As localizações dos agrupamentos são distribuídas ao redor do mundo, com cada localização normalmente próxima a múltiplos pontos de presença de ISP da camada 1. Esses agrupamentos são responsáveis por servir conteúdo estático, incluindo vídeos do YouTube [Adhikari, 2011a].
- Muitas centenas de *clusters enter deep* (veja Seção 7.2.4), com cada *cluster* localizado dentro de um ISP de acesso. Este *cluster* consiste em dezenas de servidores dentro de uma única estante. Os servidores *enter deep* realizam a divisão de TCP (veja a Seção 3.7) e servem conteúdo estático [Chen, 2011], incluindo as partes estáticas das páginas da Internet que incluem resultados de busca.

Todos esses datacenters e localizações de *clusters* de servidores estão dispostos juntos em rede com a rede privada da Google, como parte de um enorme AS (AS 15169). Quando um usuário efetua um pedido de busca, geralmente esse pedido é enviado primeiro do ISP local para o *cache* de um servidor *enter deep* próximo de onde o conteúdo estático é lido; embora fornecendo o conteúdo estático ao cliente, este *cache* próximo também encaminha a consulta pela rede privada da Google para um dos mega datacenters, de onde o resultado da busca personalizada é obtido. Para um vídeo do YouTube, o vídeo em si pode vir de um dos *caches bring home*, enquanto partes da página Web ao redor do vídeo podem vir do *cache enter deep* nas vizinhanças, e os anúncios ao redor do vídeo podem vir dos datacenters. Resumindo, exceto para os ISPs locais, os serviços na nuvem da Google são, em grande parte, fornecidos por uma infraestrutura de rede que é independente da Internet pública.

Uma vez que os seus *clusters* estejam operando, a CDN replica conteúdo através dos seus *clusters*. A CDN pode não querer pôr uma cópia de todos os vídeos em cada *cluster*, já que alguns vídeos são vistos raramente ou são populares só em alguns países. De fato, muitas CDNs não empurram vídeos para seus *clusters*, mas, em vez disso, usam uma estratégia simples de puxar o conteúdo: Se um cliente requisita um vídeo de um *cluster* que não o está armazenando, o *cluster* recupera o vídeo (de um repositório central ou de outro *cluster*) e armazena uma cópia localmente enquanto envia o fluxo contínuo para o cliente ao mesmo tempo. Similar aos *caches* de Internet (veja no Capítulo 2), quando a memória do *cluster* fica cheia, ele remove vídeos que não são frequentemente requisitados.

Operação da CDN

Tendo identificado as duas principais técnicas de implantação de uma CDN, vamos agora nos aprofundar em como uma CDN opera. Quando um navegador em um hospedeiro do usuário recebe a instrução para recuperar um vídeo específico (identificado por uma URL), a CDN tem de interceptar a requisição a fim de: (1) determinar um *cluster* de servidor de CDN apropriado para o cliente naquele momento, e (2) redirecionar a requisição do cliente para um servidor naquele *cluster*. Falaremos, em linhas gerais, como uma CDN pode determinar um *cluster* apropriado. Antes, no entanto, vamos examinar os mecanismos que são usados para interceptar e redirecionar uma requisição.

A maioria das CDNs utiliza o DNS para interceptar e redirecionar requisições; uma discussão interessante sobre esse uso do DNS pode ser vista em Vixie [2009]. Consideremos um simples exemplo para ilustrar como o DNS normalmente é envolvido. Suponha que um provedor de conteúdo, NetCinema, utiliza outra companhia de CDN, KingCDN, para distribuir seus vídeos para os seus consumidores. Nas páginas de Internet do NetCinema, cada vídeo está associado a uma URL que inclui a palavra "vídeo" e um identificador único para o vídeo em si; por exemplo Tranformers 7 pode ser associado a <http://video.netcinema.com/6Y7B23V>. Então, acontecem seis passos, como mostra a Figura 7.4:

1. O usuário visita a página da Internet no NetCinema.
2. Quando o usuário clica no link <http://video.netcinema.com/6Y7B23V>, o hospedeiro do usuário envia uma consulta de DNS para video.netcinema.com.
3. O Servidor de DNS local (LDNS) do usuário retransmite a consulta de DNS a um servidor DNS autoritativo pelo NetCinema, o qual encontra a palavra "video" no nome do hospedeiro video.netcinema.com. Para "entregar" a consulta de DNS à KingCDN, em vez de um endereço IP, o servidor de DNS autoritativo

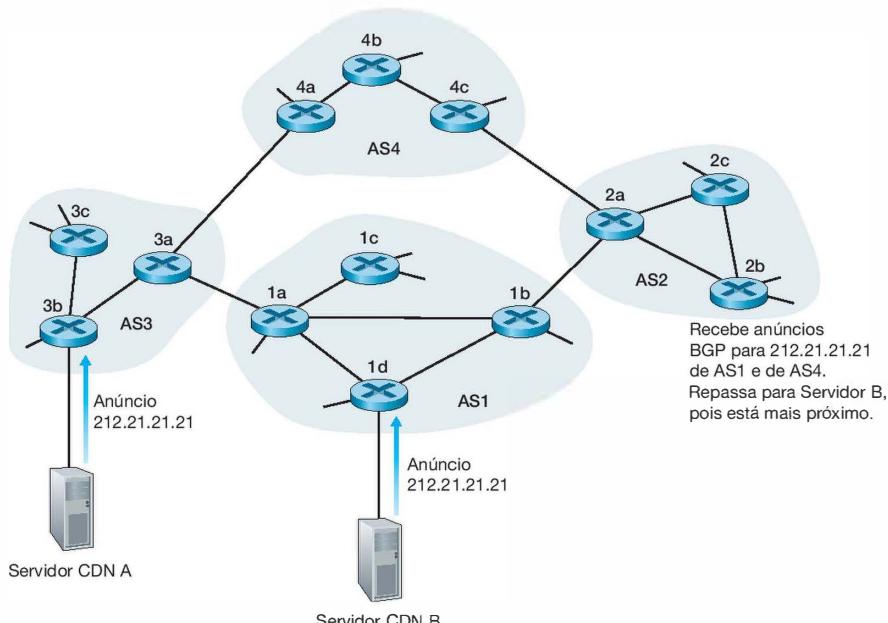
FIGURA 7.4 DNS REDIRECIONA UMA REQUISIÇÃO DO USUÁRIO PARA UM SERVIDOR DE CDN

do NetCinema retorna ao LDNS um nome de hospedeiro no domínio da KingCDN, por exemplo, a1105.kingcdn.com.

4. Deste ponto em diante, a consulta de DNS entra na infraestrutura da DNS privada da KingCDN. O LDNS do usuário, então, envia uma segunda consulta, agora para a1105.kingcdn.com, e o sistema de DNS da KingCDN por fim retorna os endereços IP de um servidor de conteúdo da KingCDN para o LDNS. Assim, é aqui, dentro do sistema de DNS da KingCDN, que é especificado o servidor de CDN de onde o cliente receberá o seu conteúdo.
5. O LDNS encaminha o endereço IP do nó de conteúdo/serviço da CDN para o hospedeiro do usuário.
6. Uma vez que o cliente obtém o endereço IP de um servidor de conteúdo da KingCDN, ele estabelece uma conexão TCP direta com o servidor que se encontra nesse endereço IP e executa uma requisição HTTP GET para obter o vídeo. Se utilizar DASH, o servidor primeiro enviará ao cliente um arquivo de manifesto com uma lista de URLs, uma para cada versão do vídeo, e o cliente irá dinamicamente selecionar trechos das diferentes versões.

Estratégias de seleção de *cluster*

No centro de qualquer distribuição de uma CDN está a **estratégia de seleção de *cluster***, isto é, um mecanismo para direcionamento dinâmico de clientes para um *cluster* de servidor ou uma central de dados dentro da CDN. Como acabamos de ver, a CDN descobre qual o endereço IP do servidor LDNS do cliente consultando o DNS do cliente. Após descobrir esse endereço IP, a CDN precisa selecionar um *cluster* apropriado baseado nesse endereço IP. As CDNs geralmente empregam estratégias próprias de seleção de *cluster*. Examinaremos agora algumas abordagens naturais, cada uma com suas vantagens e desvantagens.

Uma estratégia simples é associar o cliente ao *cluster* que está geograficamente mais próximo. Usando bases de dados de geolocalização comerciais (Quova [2012] e Max-Mind [2012], por exemplo), cada endereço IP de LDNS é mapeado para uma localização geográfica. Quando uma requisição DNS é recebida de um determinado LDNS, a CDN escolhe o *cluster* mais próximo geograficamente, isto é, o *cluster* que está a uma distância menor, em quilômetros, do LDNS. Esta solução pode funcionar razoavelmente bem para uma boa parte dos clientes [Agarwal, 2009]. No entanto, para alguns clientes, esta solução pode ter um desempenho ruim, uma vez que o *cluster* geograficamente mais próximo pode não ser o mais próximo se considerarmos o caminho percorrido pela rede. Mais ainda, existe um problema inerente a todas as abordagens baseadas em DNS, que consiste no fato de que alguns usuários finais são configurados para utilizar LDNSs remotas [Shaikh, 2001; Mao, 2002]. Nesses casos, a localização do LDNS pode ser muito longe da localização do cliente. Além disso, essa estratégia elementar ignora a variação no atraso e a largura de banda disponível dos caminhos da Internet, porque sempre associa o mesmo *cluster* a determinado cliente.

Para determinar o melhor *cluster* para um cliente baseado nas atuais condições de tráfego, as CDNs podem, como alternativa, realizar **medições em tempo real** do atraso e problemas de baixo desempenho entre seus *clusters* e clientes. Por exemplo, uma CDN pode ter cada um de seus *clusters* periodicamente enviando mensagens de verificação (por exemplo, mensagens *ping* ou consultas de DNS) para todos os LDNSs do mundo inteiro. Um obstáculo a essa técnica é o fato de que muitos LDNSs estão configurados para não responder a esse tipo de mensagem.

Uma alternativa para medir as propriedades dos caminhos, sem precisar trafegar essas mensagens estranhas, é usar as características do tráfego mais recente entre os clientes e os servidores da CDN. Por exemplo, o atraso entre um cliente e um *cluster* pode ser estimado examinando o intervalo entre um SYNACK do servidor para o cliente e o ACK do cliente para o servidor, durante a apresentação de três vias do TCP. Tais soluções, porém, requerem, de tempos em tempos, o redirecionamento de clientes para (possivelmente) *clusters* que não são os melhores, a fim de medir as propriedades dos caminhos para esses *clusters*. Embora seja necessário apenas um pequeno número de requisições para servir de mensagens de verificação, os clientes selecionados podem sofrer uma degradação de desempenho significativo quando receberem conteúdo (vídeo ou qualquer outro) [Andrews, 2002; Krishnan, 2009]. Uma alternativa para verificação do caminho *cluster*-cliente é utilizar o tráfego de consultas

DNS para medir o atraso entre clientes e *clusters*. Em especial, durante a fase de DNS (veja o Passo 4 da Figura 7.4), o LDNS do cliente pode ser ocasionalmente direcionado para servidores de autorização de DNS instalados nas várias localizações dos *clusters*, produzindo um tráfego de DNS que pode então ser medido entre o LDNS e tais localizações. Nesse método, os servidores de DNS continuam a retornar o *cluster* ideal para o cliente, de modo que a entrega de vídeos e outros objetos da Internet não seja afetada [Huang, 2010].

Uma abordagem muito diferente para combinar clientes com servidores CDN é utilizar o **IP** para qualquer membro do grupo[RFC 1546]. A ideia por trás do IP para qualquer membro do grupo é colocar os roteadores na rota da Internet dos pacotes do cliente para o *cluster* "mais próximo", como determinado pelo BGP. Especificamente, como mostrado na Figura 7.5, durante o estágio de configuração do IP para qualquer membro do grupo, a companhia de CDN associa o *mesmo* endereço IP a cada um dos seus *clusters*, e *utiliza BGP padrão* para informar esse endereço IP de cada uma das diferentes localizações de *clusters*. Quando um roteador BGP recebe múltiplos anúncios de rotas do mesmo endereço IP, trata esses anúncios como diferentes caminhos disponíveis para a mesma localização física (quando, na verdade, os anúncios são para diferentes caminhos para *diferentes* localizações físicas). Seguindo procedimentos padronizados de operação, o roteador BGP selecionará a "melhor" (por exemplo, o mais próximo, como determinado pelos contadores de salto de AS) rota para o endereço IP, de acordo com o seu mecanismo de seleção de rota local. Por exemplo, se um roteador BGP (correspondente a uma localização) está a apenas um salto de AS do roteador e todos os demais roteadores BGP (correspondentes às outras localizações) estão a dois ou mais saltos de AS, então o roteador BGP normalmente escolherá rotear pacotes para a localidade que atravessar apenas um AS (veja Seção 4.6). Depois dessa fase inicial de configuração, a CDN pode fazer o seu trabalho principal, que é distribuir conteúdo. Quando algum cliente quer assistir a algum vídeo, o DNS das CDNs retorna o endereço *para qualquer membro do grupo*, não importa onde o cliente esteja localizado. Quando o cliente envia um pacote para esse endereço IP, o pacote é roteado para o *cluster* "mais próximo", conforme determinado pelas tabelas de repasse pré-configuradas pelo BGP, como acabamos de descrever. Essa abordagem possui a vantagem de encontrar o *cluster* que é mais próximo do cliente, em vez do *cluster* que é

FIGURA 7.5 USANDO O *ANYCAST* IP PARA ROTEAR CLIENTES PARA O *CLUSTER* DA CDN MAIS PRÓXIMO

mais próximo do LDNS do cliente. No entanto, a estratégia do *anycast* IP novamente não leva em consideração a natureza dinâmica da Internet sobre escalas de tempo pequenas [Ballani, 2006].

Além das considerações relacionadas à rede, tais como atraso, perda e desempenho da largura de banda, existem muitos fatores adicionais importantes, que tratam da forma de projetar a estratégia de seleção de *cluster*. A carga nos *clusters* é um desses fatores — clientes não devem ser direcionados para *clusters* sobrecarregados. O custo de distribuição do ISP é outro fator — os *clusters* podem ser escolhidos de modo que esses ISPs específicos sejam usados para transportar o tráfego da CDN para o cliente, levando em conta as diferentes estruturas de custo nos relacionamentos contratuais entre os ISPs e os operadores dos *clusters*.

7.2.5 Estudos de caso: Netflix, YouTube e Kankan

Concluímos nossa discussão sobre vídeo de fluxo contínuo armazenado observando três implementações em larga escala de grande sucesso: Netflix, YouTube e Kankan. Veremos que todos estes sistemas utilizam diferentes métodos, empregando muitos dos princípios destacados e discutidos nesta seção.

Netflix

Gerando quase 30% de todo o tráfego de descarga de fluxo contínuo da Internet nos Estados Unidos em 2011, a Netflix tornou-se, nesse país, o provedor de serviço líder no segmento de filmes on-line e shows de TV [Sandvine, 2011]. A fim de executar rapidamente seus serviços de larga escala, tem feito uso extensivo de CDNs e serviços em nuvens de terceiros. De fato, a Netflix é um exemplo interessante de uma companhia realizando um serviço on-line em larga escala por meio de aluguel de servidores, largura de banda, armazenamento e serviços de bancos de dados de terceiros, enquanto utiliza pouca infraestrutura própria. A seguinte discussão foi adaptada de um estudo de medição — muito bem descrito — da arquitetura da Netflix [Adhikari, 2012]. Como veremos, a companhia emprega muitas das técnicas tratadas antes nesta seção, incluindo distribuição de vídeo usando CDN (na verdade, múltiplas CDNs) e HTTP de fluxo contínuo adaptativo.

A Figura 7.6 mostra a arquitetura básica da plataforma de vídeo de fluxo contínuo da Netflix. Ela possui quatro componentes principais: os servidores de registro e pagamento, a nuvem da Amazon, múltiplos provedores CDN e os clientes. Em sua infraestrutura de hardware própria, a Netflix mantém os servidores de registro e

FIGURA 7.6 PLATAFORMA DE VÍDEO DE FLUXO CONTÍNUO DA NETFLIX

pagamento, os quais permitem o registro de novas contas e capturam informações de pagamento por cartão de crédito. Com exceção destas funções básicas, a Netflix realiza seus serviços on-line implementando máquinas (ou máquinas virtuais) na nuvem da Amazon. A seguir, algumas das funcionalidades que estão na nuvem da Amazon:

- *Obtenção de conteúdo.* Antes de a Netflix poder distribuir um filme para seus clientes, é necessário obter e processar o filme. A Netflix recebe as versões principais de estúdio e as carrega para os hospedeiros na nuvem da Amazon.
- *Processamento de conteúdo.* As máquinas na nuvem da Amazon criam muitos formatos diferentes para cada filme, de acordo com uma série de tocadores de vídeo do cliente, rodando em computadores de mesa, smartphones e consoles de jogos conectados a aparelhos de televisão. Uma versão diferente é criada para cada formato e as múltiplas taxas de bits, permitindo assim que se utilize HHTP de vídeo de fluxo contínuo, usando DASH.
- *Descarregamento de versões para as CDNs.* Uma vez que todas as versões de um filme foram criadas, os hospedeiros na nuvem da Amazon descarregam as versões para as CDNs.

Para entregar filmes a seus consumidores por demanda, a Netflix faz uso extensivo da tecnologia CDN. De fato, como foi escrito em 2012, a empresa utiliza não apenas um, mas três companhias de CDN ao mesmo tempo — Akamai, Limelight e Level-3.

Tendo descrito os componentes da arquitetura da Netflix, vamos analisar mais de perto a interação entre o cliente e os vários servidores envolvidos na entrega do filme. As páginas de Internet para se navegar pela biblioteca de vídeos da Netflix usam os servidores da nuvem da Amazon. Quando o usuário seleciona um filme para "Reproduzir agora", o cliente do usuário obtém um arquivo de manifesto, proveniente também na nuvem da Amazon. Tal arquivo contém uma série de informações, incluindo uma lista ordenada de CDNs e das URLs para as diferentes versões do filme, a qual é usada para a reprodução DASH. A lista ordenada das CDNs é determinada pela Netflix, e pode mudar de uma sessão de fluxo contínuo para outra. Em geral, o cliente selecionará a CDN que está mais bem posicionada no arquivo de manifesto. Após o cliente selecionar a CDN, esta aproveita o DNS para redirecionar o cliente a um servidor específico da CDN, conforme descrito na Seção 7.2.4. O cliente e aquele servidor da CDN então interagem usando DASH. Mais especificamente, como foi descrito na Seção 7.3.2, o cliente utiliza o cabeçalho do intervalo de bytes nas mensagens de requisição HTTP GET para requisitar trechos das diferentes versões do filme. A Netflix usa trechos de cerca de 4 s [Adhikari, 2012]. Enquanto os trechos vão sendo baixados, o cliente mede a vazão recebida e executa um algoritmo para determinação da taxa, a fim de definir a qualidade do próximo trecho da requisição seguinte.

A Netflix incorpora muitos dos princípios-chave discutidos antes nesta seção, incluindo fluxo continuo adaptativo e distribuição da CDN. A empresa também gentilmente ilustra como o principal serviço de Internet, gerando quase 30% do tráfego de Internet, pode funcionar quase inteiramente em uma nuvem de terceiros e em infraestruturas de CDN de terceiros, utilizando uma infraestrutura própria tão pequena!

YouTube

Com cerca de meio bilhão de vídeos em suas bibliotecas e meio bilhão de exibições de vídeos por dia [Ding, 2011], o YouTube é indiscutivelmente o maior site da Internet no mundo para compartilhamento de vídeos. O YouTube começou seus serviços em abril de 2005 e foi adquirido pela Google em novembro de 2006. Embora o projeto e os protocolos do Google/YouTube sejam proprietários, através de muitos esforços de medição independentes, podemos ter um entendimento básico sobre como o YouTube opera [Zink, 2009; Torres, 2011; Adhikari, 2011a].

Assim como a Netflix, o YouTube faz uso extensivo da tecnologia CDN para distribuir seus vídeos [Torres, 2011]. Ao contrário da Netflix, porém, a Google não emprega CDNs de terceiros, mas, em vez disso, utiliza sua própria CDN privada para distribuir os vídeos do YouTube. A Google tem instalado *clusters* de servidores em centenas de locais diferentes. De um subconjunto de cerca de 50 desses locais, a Google distribui os vídeos do YouTube [Adhikari 2011a]. A Google usa DNS para redirecionar uma requisição do consumidor para um *cluster* específico, conforme descrito na Seção 7.2.4. Na maior parte do tempo, a estratégia de seleção de *cluster* da Google

direciona o cliente ao *cluster* para o qual o RTT entre o cliente e o *cluster* seja o menor; porém, a fim de balancear a carga através dos *clusters*, algumas vezes o cliente é direcionado (via DNS) a um *cluster* mais distante [Torres, 2011]. Além disso, se não possui o vídeo solicitado, em vez de buscá-lo de algum outro lugar e retransmiti-lo para o cliente, o *cluster* pode retornar uma mensagem de redirecionamento HTTP, desse modo redirecionando o cliente para outro *cluster* [Torres, 2011].

O YouTube emprega HTTP de fluxo contínuo, como foi descrito na Seção 7.2.2. Muitas vezes disponibiliza uma pequena quantidade de versões diferentes do vídeo, cada uma com uma taxa de bits diferente e correspondente ao nível de qualidade. Assim como ocorreu em 2011, o YouTube não utiliza fluxo contínuo adaptativo (como o DASH), mas em vez disso, requer que o usuário selecione manualmente uma versão. No intuito de economizar tanto largura de banda quanto recursos do servidor, que poderiam ser desperdiçados por reposicionamento ou por término precoce, o YouTube usa a requisição HTTP de intervalo de bytes, de modo a limitar o fluxo de dados transmitidos após uma quantidade do vídeo ter sido obtida pela pré-busca.

Alguns milhões de vídeos são enviados ao YouTube diariamente. Não apenas vídeos do YouTube são enviados do servidor ao cliente por HTTP, mas os alimentadores também enviam seus vídeos do cliente ao servidor por HTTP. O YouTube processa cada vídeo que recebe, convertendo-o para um formato de vídeo próprio e criando várias versões em diferentes taxas de bits. Esse processamento ocorre todo dentro dos datacenters da Google. Assim, ao contrário do Netflix, que executa seu serviço quase totalmente em infraestruturas de terceiros, a Google executa o serviço YouTube inteiro dentro de sua própria vasta infraestrutura de datacenters, CDN privada e rede global interconectando seus centros de dados e *clusters* de CDN. (Veja o estudo de caso sobre a infraestrutura de rede da Google, na Seção 7.2.4.)

Kankan

Acabamos de ver que, tanto nos serviços do Netflix quanto nos do YouTube, os servidores operados pelas CDNs (tanto CDNs privadas quanto de terceiros) fazem o fluxo contínuo de vídeo para seus clientes. O Netflix e o YouTube não apenas precisam pagar pelo hardware do servidor (tanto diretamente por mantê-lo, quanto indiretamente através do aluguel), mas também pela largura de banda que os servidores utilizam para distribuir os seus vídeos. Considerando a grande quantidade de serviços e de largura de banda que eles consumem, estas implementações "cliente-servidor" são extremamente custosas.

Concluímos esta seção descrevendo uma abordagem inteiramente diferente de fornecer vídeo por demanda, em larga escala, através da Internet — uma maneira que permite ao provedor de serviços reduzir significativamente sua infraestrutura e custos de largura de banda. Como você deve suspeitar, esta abordagem utiliza entrega via P2P em vez de entregas cliente-servidor (via CDNs). Entrega de vídeo P2P é usada com grande sucesso através de muitas companhias na China, incluindo a Kankan (pertence e é operada pela Xunlei), PPTV (antes era PPLive), e PPs (antes era PPstream). A Kankan, que é a líder do setor de fornecimento de vídeo por demanda P2P na China, possui mais de 20 milhões de usuários únicos visualizando seus vídeos todo mês.

Em uma análise de alto nível, o vídeo por demanda via P2P é muito parecido com a descarga de arquivos BitTorrent (discutida no Capítulo 2). Quando um par quer assistir um vídeo, entra em contato com um rastreador (que pode estar centralizado ou baseado em um par usando um DHT) para descobrir outros pares dentro do sistema que tenham uma cópia daquele vídeo. Este par então requisita trechos do arquivo de vídeo, e isto paralelamente aos outros pares que possuem o arquivo. Diferente do modo que o BitTorrent opera, porém, as requisições são preferencialmente feitas para trechos que estão para ser reproduzidos em um futuro próximo a fim de garantir uma reprodução contínua.

O projeto da Kankan utiliza um rastreador e seu próprio DHT para rastrear conteúdos. As dimensões de multidão para o conteúdo mais popular envolvem dezenas de milhares de pares, normalmente maior que as maiores multidões no BitTorrent [Dhungel, 2012]. Os protocolos da Kankan — para comunicação entre par e rastreador, entre par e DHT e entre pares — todos são proprietários. Curiosamente, para distribuir trechos de vídeos entre pares, Kankan utiliza UDP sempre que possível, levando a uma enorme quantidade de tráfego UDP dentro da Internet da China [Zhang M., 2010].

7.3 VOICE-OVER-IP

Áudio interativo em tempo real pela Internet é frequentemente denominado **telefone por Internet**, já que, da perspectiva do usuário, é semelhante ao tradicional serviço telefônico por comutação de circuitos. Também é chamado comumente por **Voice-over-IP (VoIP)**. Nesta seção descrevemos os princípios e protocolos fundamentais do VoIP. Vídeo interativo é similar em muitos aspectos ao VoIP, exceto pelo fato de que ele inclui o vídeo dos participantes assim como as suas vozes. Para que a discussão seja mantida dentro do foco e de modo concreto, enfatizaremos somente a voz nesta seção, em vez de abordar voz e vídeo combinados.

7.3.1 As limitações de um serviço IP de melhor esforço

O protocolo de camada de rede da Internet, IP, provê um serviço de melhor esforço. Em outras palavras, a Internet faz seu melhor esforço para transportar cada datagrama do remetente ao receptor o mais rapidamente possível, mas não faz nenhuma promessa sequer sobre o atraso fim a fim para um pacote individual, ou sobre o limite de perdas de pacotes. A falta dessas garantias impõe desafios significativos ao projeto de aplicações de áudio interativo em tempo real, que são bastante sensíveis ao atraso de pacotes, variação de atraso e perdas.

Nesta seção, abordaremos vários meios de o desempenho do VoIP em uma rede de melhor esforço ser aprimorado. Nosso foco será em técnicas da camada de aplicações da Internet, isto é, abordagens que não exijam quaisquer mudanças na parte principal da rede ou mesmo na camada de transporte no lado dos hospedeiros. Para manter a discussão em termos concretos, discutiremos as limitações de um serviço de IP de melhor esforço no contexto de um exemplo específico de VoIP. O emitente gera bytes a uma taxa de 8.000 bytes por segundo; a cada 20 ms o emitente agrupa esses bytes em um bloco. Um bloco e um cabeçalho especial (discutiremos sobre isso em seguida) são encapsulados em um segmento UDP, por meio de uma chamada a interface do *socket*. Sendo assim, o número de bytes do bloco é (20 ms) · (8.000 bytes/s) = 160 bytes, e um segmento de UDP é enviado a cada 20 ms.

Se cada pacote chega ao receptor com um atraso fim a fim constante, então os pacotes chegam no receptor a cada 20 ms. Nessas condições ideais, o receptor pode simplesmente reproduzir cada parte assim que ele chega. Infelizmente alguns pacotes talvez se percam e a maioria deles não possuirá o mesmo atraso fim a fim, ainda que a Internet esteja ligeiramente congestionada. Por este motivo, o receptor deve se empenhar em determinar (1) quando reproduzir uma parte e (2) o que fazer com uma parte perdida.

Perda de pacotes

Considere um dos segmentos UDPs, gerado por nossa aplicação VoIP. O segmento UDP é encapsulado por um datagrama IP. Enquanto o datagrama vagueia pela rede, ele passa por buffers (isto é, por filas) nos roteadores, aguardando para ser transmitido em enlaces de saída. É possível que um ou mais dos buffers na rota entre o remetente e o destinatário estejam lotados e não possam aceitar o datagrama IP. Nesse caso, o datagrama IP será descartado e nunca chegará à aplicação receptora.

A perda pode ser eliminada enviando os pacotes por TCP (que proporciona transferência de dados confiável), em vez de por UDP. Contudo, os mecanismos de retransmissão frequentemente são considerados inaceitáveis para aplicações interativas de áudio em tempo real, como o VoIP, porque aumentam o atraso fim a fim Bolot [1996]. Além disso, por causa do controle de congestionamento do TCP, após a perda de pacote a taxa de transmissão no remetente pode ser reduzida a uma taxa mais baixa do que a de reprodução no receptor, possivelmente ocasionando inanição no buffer. Isto pode causar um forte impacto sobre a integridade da voz no receptor. Por essas razões, quase todas as aplicações de VoIP existentes executam em UDP. Baset [2006] relata que o UDP é usado pelo Skype a não ser que o usuário esteja atrás de um NAT ou de um *firewall* que bloqueia os segmentos UDP (nesse caso TCP é usado).

Mas perder pacotes talvez não seja tão desastroso quanto se possa imaginar. Na verdade, taxas de perda de pacotes entre 1% e 20% podem ser toleradas, dependendo de como a voz é codificada e transmitida e de como a perda é ocultada no receptor. Por exemplo, o mecanismo de correção de erros de repasse (*forward error*

correction — FEC) pode ajudar a ocultar a perda de pacotes. Veremos mais adiante que, com a FEC, a informação redundante é transmitida junto com a informação original, de modo que parte dos dados originais perdidos pode ser recuperada da informação redundante. Não obstante, se um ou mais dos enlaces entre remetente e receptor estiverem fortemente congestionados e a perda de pacotes exceder 10–20% (por exemplo, em um enlace sem fio), então, na realidade, nada poderá ser feito para conseguir uma qualidade de som aceitável. Assim, fica claro que o serviço de melhor esforço tem suas limitações.

Atraso de fim a fim

Atraso de fim a fim é o acúmulo de atrasos de processamento, de transmissão e de formação de filas nos roteadores; atrasos de propagação nos enlaces e atrasos de processamento em sistemas finais. Para as aplicações de áudio altamente interativas em tempo real, como o VoIP, atrasos fim a fim menores do que 150 ms não são percebidos pelo ouvido humano; entre 150 e 400 ms podem ser aceitáveis, mas não são o ideal e os que excedem 400 ms podem atrapalhar seriamente a interatividade em conversações por voz. O lado receptor de uma aplicação de telefone por Internet em geral desconsiderará quaisquer pacotes cujos atrasos ultrapassarem determinado patamar, por exemplo, mais do que 400 ms. Assim, os pacotes cujos atrasos ultrapassem o patamar são efetivamente perdidos.

Variação de atraso do pacote

Um componente crucial do atraso fim a fim são os atrasos variáveis de fila que os pacotes sofrem nos roteadores. Por causa de tais atrasos, o tempo decorrido entre o momento em que um pacote é gerado na fonte e o momento em que é recebido no destinatário pode variar de pacote para pacote, como mostra a Figura 7.1. Esse fenômeno é denominado **variação de atraso**. Como exemplo, considere dois pacotes consecutivos em nossa aplicação de telefone por Internet. O remetente envia o segundo pacote 20 ms após ter enviado o primeiro. Mas, no receptor, o espaçamento entre eles pode ficar maior do que 20 ms. Para entender, suponha que o primeiro pacote chegue a uma fila de roteador quase vazia, mas que, exatamente antes de o segundo pacote chegar, um grande número de pacotes vindos de outras fontes chegue à mesma fila. Como o primeiro pacote sofre um pequeno atraso de fila e o segundo pacote sofre um grande atraso de fila nesse roteador, o primeiro e o segundo pacotes ficam espaçados em mais de 20 ms. O espaçamento entre pacotes consecutivos também pode ficar menor do que 20 ms. Para entender, considere novamente dois pacotes consecutivos. Suponha que o primeiro se junte ao final de uma fila com um grande número de pacotes e que o segundo chegue à fila antes dos pacotes de outras fontes. Nesse caso, nossos dois pacotes se encontram um exatamente atrás do outro na fila. Se o tempo para transmitir um pacote no enlace de saída do roteador for menor do que 20 ms, então o primeiro e o segundo pacotes ficarão espaçados por menos de 20 ms.

A situação é análoga a dirigir carros em rodovias. Suponha que você e um amigo seu, cada um dirigindo seu próprio carro, estejam viajando de San Diego a Phoenix. Suponha que você e seu amigo tenham um estilo semelhante de dirigir e mantenham a velocidade de 100 km/h, se o tráfego permitir. Por fim, imagine que seu amigo parta uma hora antes de você. Então, dependendo do tráfego, você pode chegar a Phoenix mais ou menos uma hora depois de seu amigo.

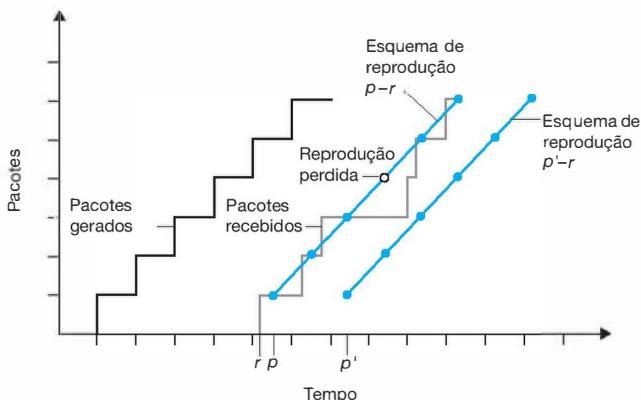
Se o receptor ignorar a presença de variação de atraso e reproduzir as porções assim que elas chegam, então a qualidade de áudio resultante poderá facilmente se tornar ininteligível no receptor. Felizmente, a variação de atraso quase sempre pode ser eliminada com a utilização de **números de sequência**, **marcas de tempo** e **atraso de reprodução**, como discutido a seguir.

7.3.2 Eliminação da variação de atraso no receptor para áudio

Para uma aplicação de VoIP, em que pacotes vão sendo gerados periodicamente, o receptor deve tentar prover reprodução sincronizada de porções de voz na presença de variação de atraso aleatório. Isto normalmente é feito combinando os dois mecanismos seguintes:

- *Preceder cada parte com uma* ***marca de tempo***. O remetente marca cada parte com o instante em que ela foi gerada.
- ***Atrasar a reprodução*** *de porções no receptor*. Como vimos em nossa discussão anterior da Figura 7.1, o atraso da reprodução das porções de áudio recebidas deve ser longo o suficiente para que a maioria dos pacotes seja recebida antes de seus tempos de reprodução programados. O atraso da reprodução pode ser fixado para todo o período de duração da sessão de áudio ou pode variar adaptativamente durante o período útil da sessão.

Discutiremos agora como esses três mecanismos, quando combinados, podem atenuar ou até eliminar os efeitos da variação de atraso. Examinaremos duas estratégias de reprodução: atraso de reprodução fixo e atraso de reprodução adaptativo.

Atraso por reprodução fixa

Com a estratégia do atraso fixo, o receptor tenta reproduzir cada parte exatamente q milissegundos após a parte ter sido gerada. Assim, se a marca de tempo de uma parte for t, o receptor reproduz a parte no tempo $t + q$, admitindo-se que a parte já tenha chegado naquele instante. Os pacotes que chegam após seus tempos de reprodução programados são descartados e considerados perdidos.

Qual é uma boa escolha para q? VoIP pode suportar atrasos de até cerca de 400 ms, embora com valores menores que q a experiência interativa resultante seja mais satisfatória. Por outro lado, caso se escolha um q muito menor do que 400 ms, muitos pacotes podem perder seus tempos de reprodução programados por causa da variação de atraso induzida pela rede. Em termos gerais, se grandes variações no atraso fim a fim forem características, será preferível usar um q grande; por outro lado, se o atraso for pequeno e as variações também forem pequenas, será preferível usar um q pequeno, talvez menor do que 150 ms.

O compromisso entre o atraso de reprodução e a perda de pacotes é ilustrado na Figura 7.7. A figura mostra os tempos em que os pacotes são gerados e reproduzidos para uma única rajada de voz. Dois atrasos de reprodução iniciais distintos são considerados. Como mostram os degraus mais à esquerda do gráfico, o remetente gera pacotes a intervalos regulares — digamos, a cada 20 ms. O primeiro pacote dessa rajada de voz é recebido no tempo r. Como mostra a figura, as chegadas dos pacotes subsequentes não são espaçadas regularmente por causa da variação de atraso da rede.

Para o primeiro esquema de reprodução, o atraso inicial está estabelecido em $p - r$. Com esse esquema, o quarto pacote não chega dentro de seu atraso de reprodução programado e o receptor o considera perdido.

FIGURA 7.7 PERDA DE PACOTES PARA DIFERENTES ATRASOS POR REPRODUÇÃO FIXA

Pelo segundo esquema, o atraso inicial de reprodução fixo está estabelecido em $p' - r$. Aqui, todos os pacotes chegam antes de seu tempo de reprodução e, portanto, não há perda.

Atraso por reprodução adaptativa

O exemplo anterior demonstra um importante compromisso entre atraso e perda que surge ao se projetarem esquemas de reprodução com atrasos por reprodução fixa. Ao se decidir por um atraso inicial de reprodução grande, a maioria dos pacotes cumprirá suas programações e, portanto, a perda será insignificante; contudo, para serviços interativos como VoIP, atrasos longos podem se tornar incômodos, se não intoleráveis. Idealmente, gostaríamos que o atraso de reprodução fosse minimizado mas que mantivesse a limitação de perda abaixo de uns poucos por cento.

A maneira natural de tratar esse compromisso é estimar o atraso de rede e sua variância e ajustar o atraso de reprodução de acordo com o resultado, no início de cada rajada de voz. Esse ajuste adaptativo de atrasos de reprodução no início das rajadas de voz fará que os períodos de silêncio no receptor sejam comprimidos e alongados; contudo, compressão e alongamento de silêncio em pequenas quantidades não são percebidos durante a fala.

Tomando Ramjee [1994] como base, escrevemos agora um algoritmo genérico que o receptor pode usar para ajustar seus atrasos de reprodução adaptativamente. Para tal, consideremos:

t_i = marca de tempo do *i*-ésimo pacote = o instante em que o pacote foi gerado pelo remetente
r_i = o instante em que o pacote *i* é recebido pelo receptor
p_i = o instante em que o pacote *i* é reproduzido no receptor

O atraso de rede fim a fim do *i*-ésimo pacote é $r_i - t_i$. Por causa da variação de atraso da rede, esse atraso variará de pacote a pacote. Seja d_i a estimativa do atraso de rede *médio* para a recepção do *i*-ésimo pacote. Essa estimativa é obtida das marcas de tempo como segue:

$$d_i = (1 - u)\, d_{i-1} + u\,(r_i - t_i)$$

sendo u uma constante fixa (por exemplo, $u = 0{,}01$). Assim, d_i é uma média ajustada dos atrasos de rede observados $r_1 - t_1, \ldots, r_i - t_i$. A estimativa atribui maior peso aos atrasos de rede verificados mais recentemente do que aos atrasos de rede ocorridos no passado distante. Esse modelo de estimativa não deve ser tão fora do comum; uma ideia semelhante é usada para estimar os tempos de viagem de ida e volta no TCP, como discutido no Capítulo 3. Seja v_i uma estimativa do desvio médio do atraso em relação ao atraso médio estimado. A estimativa também é obtida com base nas marcas de tempo:

$$v_i = (1 - u)\, v_{i-1} + u\, | r_i - t_i - d_i |$$

As estimativas d_i e v_i são calculadas para todos os pacotes recebidos, embora somente sejam usadas para determinar o ponto de reprodução para o primeiro pacote em qualquer rajada de voz.

Uma vez calculadas essas estimativas, o receptor emprega o seguinte algoritmo para a reprodução de pacotes. Se o pacote *i* for o primeiro de uma rajada de voz, seu tempo de reprodução p_i será computado como:

$$p_i = t_i + d_i + Kv_i$$

sendo K uma constante positiva (por exemplo, $K = 4$). A finalidade do termo Kv_i é estabelecer o tempo de reprodução em um futuro longínquo o suficiente, de modo que apenas uma pequena fração dos pacotes que estão chegando na rajada de voz seja perdida devido às chegadas das atrasadas. O ponto de reprodução para qualquer pacote subsequente em uma rajada de voz é computado como um deslocamento em relação ao ponto no tempo em que o primeiro pacote foi reproduzido. Em particular, seja

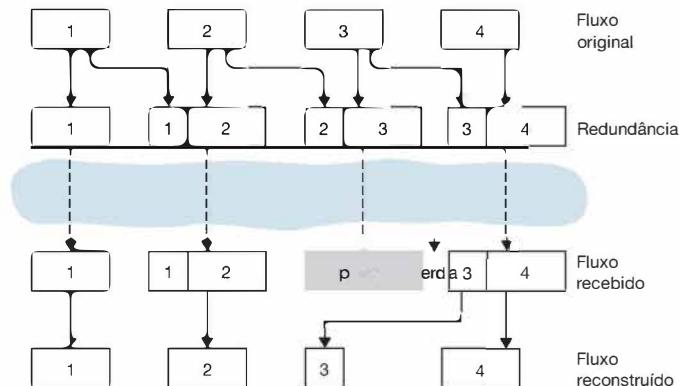
$$q_i = p_i - t_i$$

a duração do tempo decorrido desde a geração do primeiro pacote da rajada de voz até sua reprodução. Se o pacote *j* também pertencer a essa rajada de voz, ele será reproduzido no tempo

$$p_j = t_j + q_i$$

O algoritmo que acabamos de descrever tem perfeito sentido admitindo-se que o receptor possa dizer se um pacote é o primeiro na rajada de voz. Isto pode ser feito examinando o sinal de energia em cada pacote recebido.

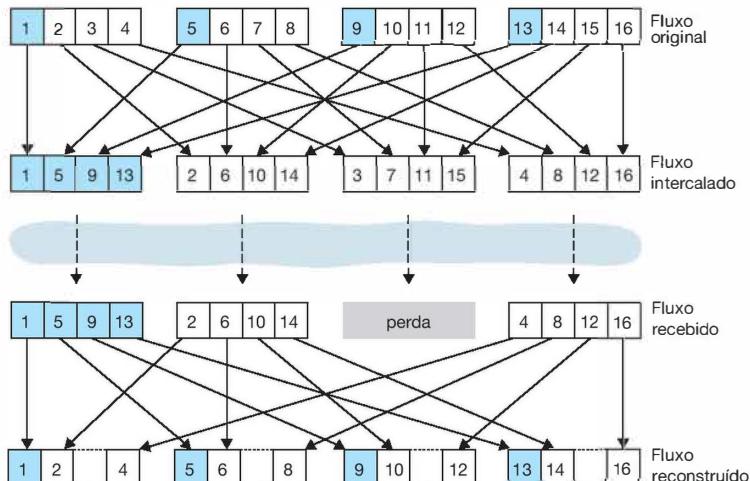
7.3.3 Recuperação de perda de pacotes

Já discutimos com algum detalhe como uma aplicação de VoIP pode lidar com a variação de atraso de pacote. Agora, descreveremos de modo breve diversos esquemas que tentam preservar uma qualidade aceitável de áudio na presença da perda de pacotes. Eles são denominados **esquemas de recuperação de perdas**. Definimos aqui a perda de pacotes em um sentido amplo: um pacote será considerado perdido se nunca chegar ao receptor ou se chegar após o tempo de reprodução programado. Nosso exemplo do VoIP servirá outra vez de contexto para a descrição de esquemas de recuperação de perdas.

Como mencionamos no início desta seção, em geral a retransmissão de pacotes perdidos não é apropriada para aplicações interativas em tempo real como VoIP. Na verdade, a retransmissão de um pacote que perdeu seu prazo de reprodução não serve para absolutamente nada. E a retransmissão de um pacote que transbordou de uma fila de roteador em geral não pode ser feita com a necessária rapidez. Por essas considerações, aplicações de VoIP frequentemente usam algum tipo de esquema de prevenção de perda. Dois desses esquemas são a **FEC** e a **intercalação**.

Forward Error Correction (FEC)

A ideia básica da FEC é adicionar informações redundantes ao fluxo de pacotes original. Ao custo de aumentar marginalmente a taxa de transmissão do áudio de fluxo, a informação redundante pode ser usada para reconstruir aproximações ou versões exatas de alguns pacotes perdidos. Esboçamos, agora, dois mecanismos de FEC segundo Bolot [1996] e Perkins [1998]. O primeiro mecanismo envia uma parte redundante codificada após cada n porções. A parte redundante é obtida por XOR as n porções originais [Shacham, 1990]. Desse modo, se qualquer pacote do grupo de $n + 1$ pacotes for perdido, o receptor poderá reconstruí-lo integralmente. Mas, se dois ou mais pacotes de um grupo forem perdidos, o receptor não poderá reconstruir os pacotes perdidos. Mantendo $n + 1$ (o tamanho do grupo) pequeno, uma grande fração dos pacotes perdidos pode ser recuperada, quando a perda não for excessiva. Contudo, quanto menor o tamanho do grupo, maior será o aumento relativo da taxa de transmissão do fluxo de áudio. Em particular, a taxa de transmissão aumenta de um fator de $1/n$; por exemplo, se $n = 3$, então a taxa de transmissão aumenta 33%. Além disso, esse esquema simples aumenta o atraso de reprodução, pois o receptor tem que esperar o término da recepção do grupo de pacotes inteiro antes de poder começar a reproduzir. Se o leitor quiser mais detalhes práticos sobre como a FEC funciona para transporte de multimídia, consulte [RFC 5109].

O segundo mecanismo de FEC é enviar um fluxo de áudio de resolução mais baixo como informação redundante. Por exemplo, o emitente pode criar um fluxo de áudio nominal e um fluxo de áudio correspondente de baixa resolução e baixa taxa de bits. (O fluxo nominal poderia ser uma codificação PCM de 64 kbits/s e o fluxo de qualidade mais baixa, uma codificação GSM de 13 kbits/s.) O fluxo de baixa taxa de bits é denominado fluxo redundante. Conforme mostra a Figura 7.8, o remetente constrói o enésimo pacote tomando a enésima parte do fluxo nominal e anexando a ele a parte de ordem $(n - 1)$ do fluxo redundante. Dessa maneira, sempre que houver perda de pacote não consecutiva, o receptor pode ocultar a perda reproduzindo a parte codificada de baixa taxa de bits que chegar com o pacote subsequente. É evidente que as porções de baixa taxa de bits oferecem qualidade mais baixa do que as porções nominais. Contudo, um fluxo no qual a maioria das porções é de alta qualidade, as partes de baixa qualidade são ocasionais e nenhuma parte perdida oferece boa qualidade geral de áudio. Note que, nesse esquema, o receptor tem de receber somente dois pacotes antes da reprodução, de modo que o aumento no atraso de reprodução é pequeno. Além disso, se a codificação de baixa taxa de bits for muito menor do que a codificação nominal, o aumento marginal da taxa de transmissão será pequeno.

FIGURA 7.8 DANDO CARONA À INFORMAÇÃO REDUNDANTE DE QUALIDADE MAIS BAIXA

Para enfrentar perdas consecutivas, podemos utilizar uma variação simples. Em vez de anexar apenas a parte de baixa taxa de bits de ordem $(n - 1)$ à enésima parte nominal, o remetente pode anexar as porções de baixa taxa de bits de ordens $(n - 1)$ e $(n - 2)$ ou anexar as porções de baixa taxa de bits de ordens $(n - 1)$ e $(n - 3)$ e assim por diante. Anexando mais porções de baixa taxa de bits a cada parte nominal, a qualidade do áudio no receptor fica aceitável para uma variedade mais ampla de ambientes de melhor esforço adversos. Por outro lado, as porções adicionais aumentam a largura de banda de transmissão e o atraso de reprodução.

Intercalação

Como uma alternativa à transmissão redundante, uma aplicação de VoIP pode enviar áudio intercalado. Como mostra a Figura 7.9, o remetente rearranja a sequência das unidades de dados de áudio antes da transmissão, para que unidades originalmente adjacentes fiquem separadas por alguma distância no fluxo transmitido.

FIGURA 7.9 ENVIO DE ÁUDIO INTERCALADO

A intercalação pode atenuar o efeito da perda de pacotes. Se, por exemplo, as unidades têm comprimento de 5 ms e as porções são de 20 ms (isto é, quatro unidades por parte), então a primeira parte pode conter unidades 1, 5, 9, 13; a segunda parte pode conter unidades 2, 6, 10, 14, e assim por diante. A Figura 7.9 mostra que a perda de um único pacote de um fluxo intercalado resulta em várias pequenas lacunas no fluxo reconstruído, em vez de em uma única lacuna grande que ocorreria com um sistema não intercalado.

A intercalação pode melhorar significativamente a qualidade percebida de um fluxo de áudio [Perkins, 1998]. Além disso, a sobrecarga é baixa. A desvantagem óbvia da intercalação é que ela aumenta a latência, o que limita seu uso em aplicações interativas como o VoIP, embora possa funcionar bem para fluxo contínuo de áudio armazenado. Uma importante vantagem da intercalação é que ela não aumenta as exigências de largura de banda de um fluxo.

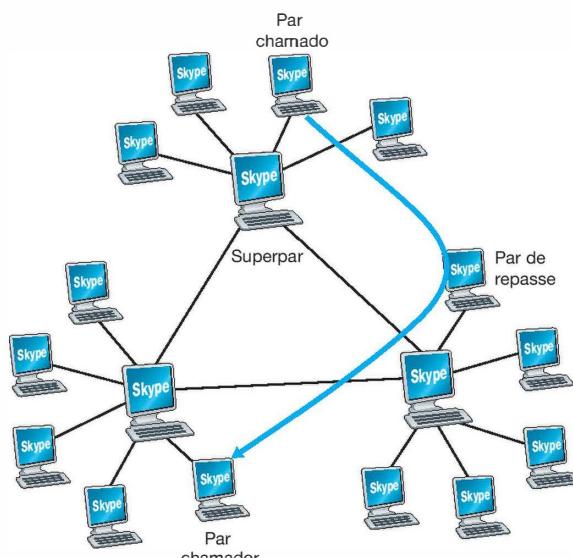
Ocultação de erro

Esquemas de recuperação baseados no receptor tentam produzir um substituto para um pacote perdido semelhante ao original. Como discutido em Perkins [1998], isso é possível desde que os sinais de áudio, em especial os de voz, exibam grandes índices de semelhança entre si dentro de períodos de tempo reduzidos. Assim, essas técnicas funcionam para taxas de perda relativamente pequenas (menos de 15%) e para pacotes pequenos (4–40 ms). Quando o comprimento da perda se aproxima do comprimento de um fonema (5–100 ms), essas técnicas causam pane, já que fonemas inteiros podem ser perdidos pelo ouvinte.

Talvez o modo mais simples de recuperação baseada no receptor seja a repetição de pacotes. Ela substitui pacotes perdidos por cópias de pacotes que chegaram imediatamente antes da perda. É de baixa complexidade computacional e funciona bastante bem. Outro modo de recuperação baseada no receptor é a interpolação, que usa áudio antes e após a perda para interpolar um pacote adequado para cobrir a perda. Ela funciona um pouco melhor do que a repetição de pacotes, mas é significativamente mais trabalhosa para processamento [Perkins, 1998].

7.3.4 Estudo de caso: VoIP com Skype

Skype é uma aplicação extremamente popular, com mais de 50 milhões de contas ativas. Além de fornecer serviço de VoIP, o Skype oferece serviços de chamadas de hospedeiro para telefone, de telefone para hospedeiro e serviços de videoconferência de hospedeiro para hospedeiro com múltiplos participantes. (Um hospedeiro, neste contexto, é qualquer dispositivo IP conectado à Internet.) O Skype foi adquirida pela Microsoft em 2011 por mais de 8 bilhões de dólares.

Como o protocolo do Skype é proprietário, e todos os controles e pacotes do Skype são criptografados, fica difícil determinar exatamente como o Skype opera. Apesar disso, a partir do site da Internet do Skype e de muitos estudos sobre métricas, pesquisadores têm compreendido cada vez melhor como, em geral, o Skype funciona [Baset, 2006; Guha, 2006; Chen, 2006; Suh, 2006; Ren, 2006; Zhang X, 2012]. Tanto para voz quanto para vídeo, os clientes têm à disposição muitos diferentes codecs, os quais são capazes de codificar a mídia em um gama de taxas e qualidades diferentes. Por exemplo, as taxas de vídeo para Skype medidas variam de 30 kbits/s para uma sessão de baixa qualidade até 1 Mbit/s para uma sessão de alta qualidade [Zhang X, 2012]. Normalmente a qualidade do áudio do Skype é melhor que a dos serviços de telefonia antigos ("POTS" — Plain Old Telephone Service), fornecida pelo sistema de telefonia convencional por fio (os codecs do Skype fazem a amostragem da voz numa velocidade de 16.000 amostras/s ou mais, o que oferece tons de mais qualidade que o POTS, que amostra a voz a 8.000 amostras/s). O Skype padronizou o uso de UDP para o envio de pacotes de áudio e vídeo. No entanto, os pacotes de controle são enviados via TCP, e pacotes de mídia são também enviados via TCP quando os *firewalls* bloqueiam fluxo contínuo de UDP. O Skype usa FEC para recuperação de perdas de fluxo contínuo tanto de voz quanto de vídeo. A aplicação cliente também adapta os fluxos contínuos de voz e vídeo que ela envia às condições atuais da rede, através de mudanças na qualidade do vídeo e *overhead* de FEC [Zhang X, 2012].

Skype usa técnicas P2P de muitas formas inovadoras, ilustrando de uma maneira bem interessante como P2P pode ser usado em aplicações que usam distribuição de conteúdo e compartilhamento de arquivos. Assim como

mensagens instantâneas, telefonia por Internet de hospedeiro para hospedeiro é, em sua natureza, uma técnica P2P, já que, na essência da aplicação, pares de usuários se comunicam entre si em tempo real. A Skype, porém, também emprega técnicas P2P para duas outras funções importantes: localização de usuário e travessia de NAT.

Como apresentado na Figura 7.10, os pares (hospedeiros) na Skype são organizados em uma rede de sobreposição hierárquica, com cada par classificado como um superpar ou um par comum. Skype mantém um índice que mapeia os nomes de usuários do Skype com seus endereços IP atuais (e números de porta). Esse índice é distribuído para todos os superpares. Quando Alice quer chamar Bob, seu aplicativo cliente Skype procura o índice distribuído para determinar o atual endereço IP de Bob. Como os protocolos Skype são proprietários, ainda não é conhecido como os mapas de índice são organizados através dos superpares, embora seja muito provável que se use alguma forma de organização de DHT.

Técnicas de P2P também são usadas nos **repasses** (*relays*) do Skype, onde são úteis para estabelecer chamadas entre hospedeiros em redes domésticas. Muitas configurações de redes domésticas fornecem acesso à Internet através de NATs, como foi discutido no Capítulo 4. Lembre-se que um NAT impede que um hospedeiro de fora da rede doméstica inicie uma conexão com um hospedeiro dentro da rede doméstica. Mas se *os dois* que desejam se comunicar usarem NATs, então existe um problema — nenhum deles poderá aceitar chamadas iniciadas pelo outro, o que tornaria a chamada impossível de ser realizada. O uso inteligente de superpares e retransmissões pode muito bem resolver este problema. Suponha que, quando Alice se conecta, ela é associada a um superpar que não está usando NAT e inicia uma sessão com esse superpar. (Como Alice está iniciando a sessão, o seu NAT permite essa sessão.) Essa sessão possibilita que Alice e o seu superpar troquem mensagens de controle. O mesmo acontece para Bob quando ele se conecta. Então, quando Alice quiser chamar Bob, ela informa o superpar dela, que por sua vez informa o superpar de Bob, que então informa Bob da chamada de Alice para ele. Se Bob aceitar, os dois superpares selecionam um terceiro superpar que não esteja usando NAT — o par de retransmissão — cujo trabalho será retransmitir dados entre Alice e Bob. Os superpares de Alice e Bob então são orientados a iniciar uma sessão com o retransmissor. Como mostrado na Figura 7.10, Alice então envia pacotes de voz para o retransmissor pela conexão estabelecida entre eles (a qual foi iniciada por Alice) e o retransmissor então encaminha esses pacotes pela conexão do retransmissor com Bob (o qual foi iniciado por Bob); pacotes de

FIGURA 7.10 PARES NO SKYPE

Bob para Alice fluem por estes mesmos retransmissores ao contrário. E *voilà*! — Bob e Alice estabelecem uma conexão fim a fim, embora nenhum dos dois possa acessar uma sessão originada de fora da rede doméstica.

Até agora nossa discussão sobre o Skype tem sido focada nas chamadas envolvendo duas pessoas. Agora, vamos examinar chamadas de audioconferência com mais de um participante. Com $N > 2$ participantes, se cada usuário enviar uma cópia do seu fluxo contínuo de áudio para cada um dos outros $N - 1$ usuários, então um total de $N(N - 1)$ fluxos contínuos de áudio precisará ser enviado pela rede para suportar a audioconferência. A fim de reduzir o uso de sua largura de banda, Skype emprega uma técnica de distribuição mais inteligente. Especificamente, cada usuário envia seu fluxo continuo de áudio para o promotor da conferência. O promotor então combina os fluxos contínuos de áudio em um fluxo contínuo (basicamente adicionando todos os sinais de áudio juntos) e então envia uma cópia de cada fluxo contínuo combinado para cada um dos outros $N - 1$ participantes. Desta maneira, o número de fluxos contínuos é reduzido para $2(N - 1)$. Para conversações simples de vídeo, envolvendo apenas duas pessoas, o Skype envia a chamada par a par, a menos que a travessia do NAT seja requerida, e neste caso a chamada é retransmitida através do par que não utiliza NAT, como descrito antes. Para uma chamada de videoconferência envolvendo $N > 2$ participantes, pela natureza do meio em que é transmitido o vídeo, o Skype não compõe a chamada em um fluxo contínuo em uma localidade e então redistribui o fluxo contínuo para todos os participantes, como faz para chamadas de voz. Em vez disso, cada participante do fluxo contínuo de vídeo é direcionado para um *cluster* de servidor (localizado na Estônia, desde 2011), o qual por sua vez redireciona para cada participante os $N - 1$ fluxos contínuos dos outros $N - 1$ participantes [Zhang X, 2012]. Você pode estar se perguntando por que cada participante envia uma cópia para o servidor em vez de direcionar uma cópia do seu fluxo contínuo para cada um dos outros $N - 1$ participantes. De fato, para ambas as abordagens, $N(N - 1)$ fluxos contínuos de vídeo estão sendo coletivamente recebidos pelos N participantes na conferência. A razão é a seguinte: como as larguras de banda dos enlaces *upstream* são significativamente menores que as larguras de banda dos enlaces de *downstream* na maioria dos enlaces de acesso, os enlaces de *upstream* podem não conseguir suportar os $N - 1$ fluxos contínuos com a abordagem P2P.

Sistemas VoIP como Skype, QQ e Google Talk introduzem novas preocupações referentes à privacidade. Especificamente, quando Alice e Bob se comunicam via VoIP, Alice pode descobrir o endereço IP de Bob e então usar serviços de geolocalização [MaxMind, 2012; Quova, 2012] para determinar o local atual de Bob e seu ISP (por exemplo, seu ISP no trabalho ou em casa). Na verdade, com Skype é possível para Alice bloquear a transmissão de determinados pacotes durante o estabelecimento da chamada, e assim ela obtém o endereço IP atual de Bob, digamos, a cada hora, sem que ele saiba que está sendo monitorado e sem estar na lista de contatos de Bob. Além disso, o endereço IP descoberto pelo Skype pode ser correlacionado com o endereço IP encontrado no BitTorrent, e então Alice pode acessar os arquivos que Bob está baixando [LeBlond, 2011]. Mais ainda, é possível decodificar em parte uma chamada Skype, fazendo uma análise do tamanho dos pacotes em um fluxo contínuo [White, 2011].

7.4 PROTOCOLOS PARA APLICAÇÕES INTERATIVAS EM TEMPO REAL

Aplicações interativas em tempo real, incluindo VoIP e videoconferência, são atraentes e muito populares. Portanto, não é surpresa que órgãos de padrões, tais como IETF e ITU, tenham se ocupado durante tantos anos (e ainda se ocupem!) com a produção de padrões para essa classe de aplicações. Tendo à mão padrões apropriados para aplicações interativas em tempo real, empresas independentes poderão criar novos produtos atraentes que interagem uns com os outros. Nesta seção estudamos RTP e SIP para aplicações interativas em tempo real. Todos os três conjuntos de padrões estão sendo implementados amplamente em produtos do setor.

7.4.1 Protocolo de tempo real (RTP)

Na seção anterior, aprendemos que o lado remetente de uma aplicação de VoIP anexa campos de cabeçalho aos trechos de áudio/vídeo antes de passá-los à camada de transporte. Esses campos contêm números de sequência

e marcas de tempo. Já que a maioria das aplicações de rede multimídia pode usar números de sequência e marcas de tempo, é conveniente ter uma estrutura de pacote padronizada que inclua campos para dados de áudio/vídeo, números de sequência e marcas de tempo, bem como outros campos potencialmente úteis. O RTP, definido no RFC 3550, é um padrão desse tipo. Ele pode ser usado para transportar formatos comuns como PCM, ACC e MP3 para som e MPEG e H.263 para vídeo. O protocolo também pode ser usado para transportar formatos proprietários de som e de vídeo. Hoje, o RTP é amplamente implementado em muitos produtos e protótipos de pesquisa. Também é complementar a outros importantes protocolos interativos de tempo real, entre eles SIP.

Nesta seção, faremos uma introdução ao RTP. Também aconselhamos o leitor a visitar o site de Henning Schulzrinne [Schulzrinne-RTP, 2012], que trata do RTP e traz uma profusão de informações sobre o assunto. O leitor também poderá visitar o site da RAT [RAT, 2012], que documenta uma aplicação de VoIP que usa RTP.

Fundamentos de RTP

O RTP comumente roda sobre UDP. O lado remetente encapsula uma parte de mídia dentro de um pacote RTP, em seguida encapsula o pacote em um segmento UDP, e então passa o segmento para o IP. O lado receptor extrai o pacote RTP do segmento UDP, em seguida extrai a parte de mídia do pacote RTP e então passa a parte para o transdutor para decodificação e apresentação.

Como exemplo, considere a utilização do RTP para transportar voz. Suponha que a origem de voz esteja codificada (isto é, amostrada, quantizada e digitalizada) em PCM a 64 kbits/s. Suponha também que a aplicação colete os dados codificados em trechos de 20 milissegundos, isto é, 160 bytes por trecho. O lado remetente precede cada parte dos dados de áudio com um **cabeçalho RTP** que contém o tipo de codificação de áudio, um número de sequência e uma marca de tempo. O tamanho do cabeçalho RTP é normalmente 12 bytes. A parte de áudio, junto com o cabeçalho RTP, forma o **pacote RTP**. O pacote RTP é, então, enviado para dentro da interface de *socket* UDP. No lado receptor, a aplicação recebe o pacote RTP da interface do seu *socket*. A aplicação extrai a parte de áudio do pacote RTP e usa os campos de cabeçalho do pacote RTP para decodificar e reproduzir adequadamente a parte de áudio.

Se uma aplicação incorporar o RTP — em vez de um esquema proprietário para fornecer tipo de carga útil, número de sequência ou marcas de tempo —, ela interagirá mais facilmente com as outras aplicações de rede multimídia. Por exemplo, se duas empresas diferentes desenvolvem um software de VoIP e ambas incorporam o RTP a seu produto, pode haver alguma chance de um usuário que estiver usando um desses produtos conseguir se comunicar com alguém que estiver usando o outro produto de VoIP. Na Seção 7.4.2, veremos que o RTP é muito utilizado em conjunto com os padrões de telefonia por Internet.

Devemos enfatizar que o RTP não fornece nenhum mecanismo que assegure a entrega de dados a tempo nem fornece outras garantias de qualidade de serviço (QoS); ele não garante nem mesmo a entrega dos pacotes nem impede a entrega de pacotes fora de ordem. Na verdade, o encapsulamento realizado pelo RTP é visto somente nos sistemas finais. Os roteadores não distinguem datagramas IP que carregam pacotes RTP de datagramas IP que não o fazem.

O RTP permite que seja atribuído a cada origem (por exemplo, uma câmera ou um microfone) seu próprio fluxo independente de pacotes RTP. Por exemplo, para uma videoconferência entre dois participantes, quatro fluxos RTP podem ser abertos — dois para transmitir o áudio (um em cada direção) e dois para transmitir o vídeo (um em cada direção). Contudo, muitas técnicas de codificação populares — incluindo MPEG1 e o MPEG2 — conjugam áudio e vídeo em um único fluxo durante o processo de codificação. Quando áudio e vídeo são conjugados pelo codificador, somente um fluxo RTP é gerado em cada direção.

Pacotes RTP não são limitados às aplicações *individuais*. Eles podem também ser enviados por árvores *para um grupo* um-para-muitos e muitos-para-muitos. Para uma sessão *para um grupo* muitos-para-muitos, todos os remetentes e origens da sessão em geral usam o mesmo grupo para enviar seus fluxos RTP. Fluxos para um grupo RTP que formam um conjunto, como fluxos de áudio e vídeo que emanam de vários remetentes em uma aplicação de videoconferência, pertencem a uma **sessão RTP**.

Campos do cabeçalho do pacote RTP

Como mostra a Figura 7.11, os quatro principais campos de cabeçalho do pacote RTP são os campos de tipo da carga útil, número de sequência, marca de tempo e os campos identificadores da origem.

O campo de tipo de carga útil do pacote RTP tem 7 bits de comprimento. Para um fluxo de áudio, o campo de tipo de carga útil serve para indicar o tipo de codificação de áudio (por exemplo, PCM, modulação delta adaptativa, codificação por previsão linear) que está sendo usado. Se um remetente decidir mudar a codificação no meio de uma sessão, ele poderá informar a mudança ao receptor por meio desse campo de tipo de carga útil. O remetente pode querer mudar o código para aumentar a qualidade do áudio ou para reduzir a taxa de bits do fluxo RTP. A Tabela 7.2 apresenta alguns dos tipos de carga útil de áudio suportados pelo RTP.

Para um fluxo de vídeo, o tipo de carga útil é usado para indicar o tipo de codificação de vídeo (por exemplo, Motion JPEG, MPEG1, MPEG2, H.261). Novamente, o remetente pode mudar a codificação de vídeo durante a sessão. A Tabela 7.3 apresenta alguns tipos de carga útil de vídeo suportados pelo RTP. Os outros campos importantes são:

- *Campo do número de sequência.* O campo do número de sequência tem comprimento de 16 bits. O número de sequência é incrementado de uma unidade a cada pacote RTP enviado e pode ser usado pelo receptor para detectar perda de pacotes e restaurar a sequência de pacotes. Por exemplo, se o lado receptor da aplicação receber um fluxo de pacotes RTP com uma lacuna entre os números de sequência 86 e 89, então o receptor saberá que os pacotes 87 e 88 estão faltando. O receptor poderá, então, tentar ocultar os dados perdidos.
- *Campo de marca de tempo.* O campo de marca de tempo tem 32 bits de comprimento. Ele reflete o instante da amostragem do primeiro byte no pacote RTP. Como vimos na seção anterior, o receptor pode usar marcas de tempo para eliminar a variação de atraso dos pacotes introduzida na rede e fornecer recepção

FIGURA 7.11 PARES NO SKYPE

Tipo de carga útil	Número de sequência	Marca de tempo	Identificador de sincronização da origem	Campos variados

TABELA 7.2 TIPOS DE CARGA ÚTIL DE ÁUDIO SUPORTADOS PELO RTP

Número do tipo de carga útil	Formato de áudio	Taxa de amostragem	Vazão
0	PCM μ-law	8 kHz	64 kbits/s
1	1016	8 kHz	4,8 kbits/s
3	GSM	8 kHz	13 kbits/s
7	LPC	8 kHz	2,4 kbits/s
9	G.722	16 kHz	48-64 kbits/s
14	Áudio MPEG	90 kHz	—
15	G.728	8 kHz	16 kbits/s

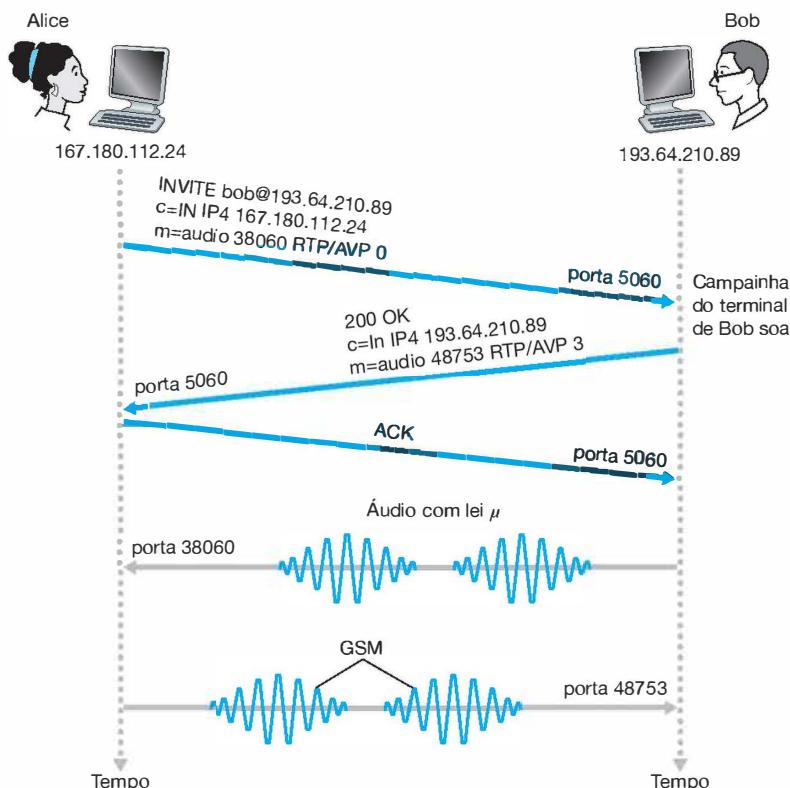
TABELA 7.3 ALGUNS TIPOS DE CARGA ÚTIL DE VÍDEO SUPORTADOS PELO RTP

Número do tipo de carga útil	Formato de vídeo
26	Motion JPEG
31	H.261
32	Vídeo MPEG 1
33	Vídeo MPEG 2

sincronizada no receptor. A marca de tempo é derivada de um relógio de amostragem no remetente. Como exemplo, para áudio, o relógio de marca de tempo é incrementado de uma unidade para cada período de amostragem (por exemplo, a cada 125 μs para um relógio de amostragem de 8 kHz); se a aplicação de áudio gerar porções consistindo em 160 amostras codificadas, a marca de tempo aumentará em 160 para cada pacote RTP enquanto a origem estiver ativa. O relógio de marca de tempo continua a aumentar a uma taxa constante, mesmo que a origem esteja inativa.

- *Identificador de sincronização da origem* (*synchronization source identifier — SSRC*). O campo SSRC tem 32 bits de comprimento e identifica a origem do fluxo RTP. Em geral, cada origem de uma sessão RTP tem um SSRC distinto. O SSRC não é o endereço IP do remetente, mas um número atribuído aleatoriamente pela origem quando um novo fluxo é iniciado. A probabilidade de que seja atribuído o mesmo SSRC a dois fluxos é muito pequena. Se isso acontecer, as duas origens escolherão novos valores SSRC.

7.4.2 SIP

O Protocolo de Inicialização de Sessão (*Session Initiation Protocol* — SIP), definido em [RFC 3261; RFC 5411], é um protocolo aberto e simples, que faz o seguinte:

- Provê mecanismos para estabelecer chamadas entre dois interlocutores por uma rede IP. Permite que quem chama avise ao que é chamado que quer iniciar uma chamada. Permite que os participantes concordem com a codificação da mídia. E também permite que encerrem as chamadas.
- Provê mecanismos que permitem a quem chama determinar o endereço IP atual de quem é chamado. Os usuários não têm um endereço IP único, fixo, porque podem receber endereços dinamicamente (usando DHCP) e porque podem ter vários equipamentos IP, cada um com um endereço IP diferente.
- Provê mecanismos para gerenciamento de chamadas, tais como adicionar novos fluxos de mídia, mudar a codificação, convidar outros participantes, tudo durante a chamada, e ainda transferir e segurar chamadas.

Estabelecendo uma chamada para um endereço IP conhecido

Para entender a essência do SIP, é melhor examinar um exemplo concreto. Nesse exemplo, Alice está trabalhando em seu computador e quer chamar Bob, que também está trabalhando no computador dele. Ambos os PCs estão equipados com software baseado em SIP para fazer e receber chamadas telefônicas. Nesse exemplo inicial, admitiremos que Alice conhece o endereço IP do PC de Bob. A Figura 7.12 ilustra o processo de estabelecimento de chamada do SIP.

Na Figura 7.12, vemos que uma sessão SIP começa quando Alice envia a Bob uma mensagem INVITE, que é parecida com uma mensagem de requisição HTTP. Essa mensagem é enviada por UDP à porta bem conhecida 5060, que é a porta do SIP. (Mensagens SIP também podem ser enviadas por TCP.) A mensagem INVITE inclui um identificador para Bob (bob@193.64.210.89), uma indicação do endereço IP atual de Alice, uma indicação de que Alice deseja receber áudio, o qual deve ser codificado em formato AVP 0 (PCM codificado com lei μ) e encapsulado em RTP e uma indicação de que ela quer receber os pacotes RTP na porta 38060. Após receber a mensagem INVITE de Alice, Bob envia uma mensagem de resposta SIP, que é parecida com uma mensagem de resposta HTTP. A mensagem de resposta SIP também é enviada à porta SIP 5060. A resposta de Bob inclui um 200 OK, bem como uma indicação de seu endereço IP, o código e o empacotamento que deseja para recepção e seu número de porta para a qual os pacotes de áudio devem ser enviados. Note que, neste exemplo, Alice e Bob vão usar mecanismos diferentes de codificação de áudio: Alice deve codificar seu áudio com GSM, ao passo que Bob deve fazê-lo com lei μ do PCM. Após receber a resposta de Bob, Alice lhe envia uma mensagem SIP de reconhecimento (ACK). Depois dessa transação SIP, Bob e Alice podem conversar. (Para melhor visualização, a Figura 7.12 mostra Alice falando depois de Bob, mas, na verdade, eles falariam ao mesmo tempo.) Bob codificará

FIGURA 7.12 ESTABELECIMENTO DE CHAMADA SIP QUANDO ALICE CONHECE O ENDEREÇO IP DE BOB

e empacotará o áudio como solicitado e enviará os pacotes de áudio para a porta número 38060 no endereço IP 167.180.112.24. Alice também codificará e empacotará o áudio como solicitado e enviará os pacotes de áudio para a porta número 48753 no endereço IP 193.64.210.89.

Com esse exemplo simples, aprendemos várias características fundamentais do SIP. Primeiro, o SIP é um protocolo "fora da banda": suas mensagens são enviadas e recebidas em portas diferentes das utilizadas para enviar e receber os dados da mídia. Segundo, as próprias mensagens SIP podem ser lidas em ASCII e são parecidas com mensagens HTTP. Terceiro, o SIP requer que todas as mensagens sejam reconhecidas, portanto, pode executar sobre UDP ou TCP.

Neste exemplo, consideremos o que aconteceria se Bob não tivesse um codec PCM μ-law para codificar áudio. Nesse caso, em vez de responder com 200 OK, Bob responderia com um 600 Not Acceptable e apresentaria na mensagem uma lista com todos os codecs que ele pode usar. Então, Alice escolheria um dos codecs da lista e enviaria outra mensagem INVITE, agora anunciando o codec escolhido. Bob também poderia apenas rejeitar a chamada enviando um dos muitos códigos de rejeição de resposta possíveis. (Há muitos desses códigos, entre eles: "ocupado" (*busy*), "encerrado" (*gone*), "pagamento solicitado" (*payment required*) e "proibido" (*forbidden*).)

Endereços SIP

No exemplo anterior, o endereço SIP de Bob é sip:bob@193.64.210.89. Contudo, esperamos que muitos endereços SIP — se não a maioria — sejam parecidos com endereços de e-mail. Por exemplo, o endereço de Bob

poderia ser sip:bob@domain.com. Quando o dispositivo SIP de Alice enviasse uma mensagem INVITE, esta incluiria esse endereço semelhante a um endereço de e-mail; a infraestrutura SIP então rotearia a mensagem ao dispositivo IP que Bob está usando no momento (como discutiremos mais adiante). Outras formas possíveis para o endereço SIP seriam o número de telefone legado de Bob ou simplesmente seu nome próprio, ou seu primeiro ou segundo sobrenome (admitindo-se que sejam únicos).

Uma característica interessante de endereços SIP é que eles podem ser incluídos em páginas Web, exatamente como endereços de e-mail são incluídos nessas páginas junto com o URL "*mail to*". Por exemplo, suponha que Bob tem uma *home page* pessoal e quer fornecer um meio para que os visitantes da página entrem e o contatem. Ele poderia, então, simplesmente incluir o URL sip:bob@domain.com. Quando o visitante clicar sobre o URL, a aplicação SIP no dispositivo do visitante será iniciada e uma mensagem INVITE será enviada a Bob.

Mensagens SIP

Nesta curta introdução ao SIP, não abordaremos todos os tipos e cabeçalhos de mensagens. Em lugar disso, examinaremos brevemente a mensagem SIP INVITE, junto com algumas linhas de cabeçalho comuns. Mais uma vez, vamos supor que Alice queira iniciar uma chamada VoIP com Bob e, agora, ela conhece só o endereço SIP de Bob, bob@domain.com, e não o endereço IP do dispositivo que Bob está usando. Então, sua mensagem poderia ser algo parecido com:

```
INVITE sip:bob@domain.com SIP/2.0
Via: SIP/2.0/UDP 167.180.112.24
From: sip:alice@hereway.com
To: sip:bob@domain.com
Call-ID: a2e3a@pigeon.hereway.com
Content-Type: application/sdp
Content-Length: 885

c=IN IP4 167.180.112.24
m=audio 38060 RTP/AVP 0
```

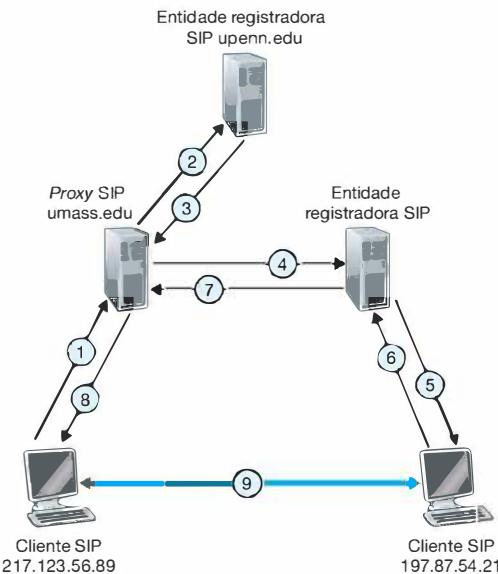
A linha INVITE inclui a versão do SIP, assim como uma mensagem de requisição HTTP. Sempre que uma mensagem SIP passa por um dispositivo SIP (incluindo o que origina a mensagem), ele anexa um cabeçalho Via, que indica o endereço IP do dispositivo. (Veremos, em breve, que uma mensagem INVITE típica passa por muitos dispositivos SIP antes de chegar à aplicação SIP do usuário que foi chamado.) Semelhante a uma mensagem de e-mail, a mensagem SIP inclui uma linha de cabeçalho From e uma linha de cabeçalho To. Inclui também um Call-ID (identificador de chamadas), que identifica a chamada exclusivamente (semelhante à mensagem-ID no e-mail). Inclui uma linha de cabeçalho Content-Type (tipo de conteúdo), que define o formato usado para descrever o conteúdo da mensagem SIP. Inclui também uma linha de cabeçalho Content-Lenght (tamanho de conteúdo), que indica o comprimento em bytes do conteúdo na mensagem. Por fim, após um *carriage return* e um *line feed*, a mensagem contém o corpo de informação. Nesse caso, o conteúdo fornece informações sobre o endereço IP de Alice e como ela quer receber o áudio.

Tradução de nome e localização de usuário

No exemplo da Figura 7.12, admitimos que o dispositivo SIP de Alice conhecia o endereço IP onde Bob podia ser contatado. Mas essa premissa é pouco realista, não só porque muitas vezes os endereços IP são atribuídos dinamicamente com DHCP, mas também porque Bob pode ter vários dispositivos IP (por exemplo, dispositivos diferentes em casa, no trabalho e no carro). Portanto, agora vamos supor que Alice conhece somente o endereço de e-mail de Bob, bob@domain.com, e que esse mesmo endereço é usado para chamadas SIP. Nesse caso, Alice precisa obter o endereço IP do dispositivo que o usuário bob@domain.com está usando no momento. Para descobri-lo, Alice cria uma mensagem INVITE que começa com INVITE bob@domain.

com SIP/2.0, e a envia a um ***proxy* SIP**. O *proxy* responderá com uma resposta SIP que poderá incluir o endereço IP do dispositivo que bob@domain.com está usando naquele momento. Como alternativa, a resposta poderia incluir o endereço IP da caixa postal de voz de Bob, ou poderia incluir um URL de uma página Web (que diz "Bob está dormindo. Não perturbe!"). Além disso, o resultado devolvido pelo *proxy* poderia depender de quem chama: se a chamada vier da sogra de Bob, ele poderia responder com o URL que aponta para a página do "estou dormindo"!

Agora, você provavelmente está imaginando como o servidor *proxy* pode determinar o endereço IP atual para bob@domain.com. Para responder a essa pergunta, é preciso dizer algumas palavras sobre um outro dispositivo SIP, a **entidade registradora SIP**. Cada usuário SIP tem uma entidade registradora associada. Sempre que um usuário lança uma aplicação SIP em um dispositivo, ela envia uma mensagem de registro SIP à entidade registradora, informando seu endereço IP atual. Por exemplo, quando Bob lançasse sua aplicação SIP em seu PDA, a aplicação enviaria uma mensagem semelhante a esta:

```
REGISTER sip:domain.com SIP/2.0
Via: SIP/2.0/UDP 193.64.210.89
From: sip:bob@domain.com
To: sip:bob@domain.com
Expires: 3600
```

A entidade registradora monitora o endereço IP atual de Bob. Sempre que ele mudar para um novo dispositivo SIP, o novo dispositivo enviará uma mensagem de registro, indicando o novo endereço IP. Além disso, se Bob permanecer no mesmo dispositivo durante um longo período de tempo, o dispositivo enviará mensagens de confirmação de registro, indicando que o endereço IP mais recentemente enviado ainda é válido. (No exemplo anterior, é preciso enviar mensagens de confirmação a cada 3.600 segundos para manter o endereço no servidor da entidade registradora.) É importante observar que a entidade registradora é semelhante a um servidor de nomes DNS com autoridade. O servidor DNS traduz nomes de hospedeiros fixos para endereços IP fixos; a entidade registradora SIP traduz identificadores fixos de pessoas (por exemplo, bob@domain.com) para endereços IP dinâmicos. Muitas vezes, as entidades registradoras SIP e os *proxies* SIP são executados no mesmo hospedeiro.

Agora vamos examinar como o servidor *proxy* SIP de Alice obtém o endereço IP atual de Bob. Vimos, na discussão anterior, que o servidor *proxy* precisa apenas transmitir a mensagem INVITE de Alice à entidade registradora/*proxy* de Bob. Então, a registradora/*proxy* poderia transmitir a mensagem ao dispositivo SIP atual de Bob. Por fim, como agora Bob já recebeu a mensagem INVITE de Alice, ele pode enviar a ela uma resposta SIP.

Como exemplo, considere a Figura 7.13, na qual jim@umass.edu, que está trabalhando em 217.123.56.89, quer iniciar uma sessão de voz sobre IP (VoIP) com keith@upenn.edu, trabalhando em 197.87.54.21. São obedecidas as seguintes etapas: (1) Jim envia uma mensagem INVITE ao *proxy* SIP de umass. (2) O *proxy* faz uma consulta DNS para a entidade registradora SIP de upenn.edu (que não é mostrada no diagrama) e então envia a mensagem ao servidor da registradora. (3) Como keith@upenn.edu não está mais registrado na entidade registradora de upenn, esta envia uma resposta de redirecionamento, indicando que é preciso tentar keith@eurecom.fr. (4) O *proxy* de umass envia uma mensagem INVITE à registradora SIP de eurecom. (5) A registradora de eurecom conhece o endereço IP de keith@eurecom.fr e repassa a mensagem INVITE ao hospedeiro 197.87.54.21, que está executando o cliente SIP de Keith. (6–8) Uma resposta SIP é devolvida por meio de registradoras/*proxies* ao cliente SIP em 217.123.56.89. (9) A mídia é enviada diretamente entre os dois clientes. (Também há uma mensagem de reconhecimento SIP, que não é mostrada.)

Nossa discussão sobre o SIP focalizou a inicialização de chamadas para chamadas de voz. Como o SIP é um protocolo de sinalização para inicializar e encerrar chamadas em geral, ele pode ser usado para chamadas de videoconferência, bem como para sessões de texto. Na verdade, o SIP tornou-se um componente fundamental em muitas aplicações de mensagem instantânea. Os leitores que quiserem saber mais sobre esse protocolo devem visitar o site de Henning Schulzrinne [Schulzrinne-SIP, 2012]. Em particular, nesse site você encontrará software de código-fonte aberto para clientes e servidores SIP [SIP Software, 2012].

FIGURA 7.13 INICIALIZAÇÃO DE SESSÃO ENVOLVENDO *PROXIES* E ENTIDADES REGISTRADORAS SIP

7.5 SUPORTE DE REDE PARA MULTIMÍDIA

Nas seções de 7.2 a 7.4, aprendemos como os mecanismos em nível de aplicação, como buffer do cliente, pré-busca, adaptação da qualidade da mídia à largura de banda disponível, reprodução adaptativa e técnicas de mitigação de perda, podem ser usados pelas aplicações de multimídia para melhorar o desempenho de uma aplicação de multimídia. Também aprendemos como as redes de distribuição de conteúdo e as redes de sobreposição P2P podem ser usadas para fornecer uma técnica *em nível de sistema* para entregar conteúdo de multimídia. Essas técnicas e abordagens são todas criadas para uso na Internet de melhor esforço de hoje. Na verdade, elas estão em uso exatamente porque a Internet oferece apenas uma única classe de serviço de melhor esforço. Mas, como projetistas de redes de computadores, não podemos evitar a pergunta sobre se a *rede* (em vez das aplicações ou apenas da infraestrutura em nível de aplicação) poderia oferecer mecanismos para dar suporte à entrega de conteúdo de multimídia. Como veremos em breve, a resposta é, naturalmente, "sim"! Mas também vimos que diversos desses novos mecanismos de rede ainda precisam ser colocados em prática. Talvez isso se deva a sua complexidade e ao fato de que as técnicas em nível de aplicação, junto com o serviço de melhor esforço e recursos de rede corretamente dimensionados (por exemplo, largura de banda), podem de fato oferecer um serviço de entrega de multimídia de fim a fim "bom o bastante" (mesmo que não seja perfeito).

A Tabela 7.4 resume três técnicas gerais para oferecer suporte em nível de rede para aplicações de multimídia.

- *Obtendo o melhor do serviço de melhor esforço.* Os mecanismos em nível de aplicação e infraestrutura que estudamos nas seções de 7.2 a 7.4 podem ser usados com sucesso em uma rede bem dimensionada, onde a perda de pacotes e o atraso excessivo de fim a fim raramente ocorrem. Quando são previstos aumentos de demanda, os ISPs empregam largura de banda e capacidade de comutação adicionais para continuar a garantir um desempenho satisfatório com atraso e perda de pacotes [Huang, 2005]. Discutiremos esse **dimensionamento de rede** melhor na Seção 7.5.1.
- *Serviço diferenciado.* Desde os primeiros dias da Internet, tem sido previsto que diferentes tipos de tráfego (por exemplo, conforme indicado no campo de tipo de serviço do cabeçalho de pacote IPv4) poderiam ser fornecidos com classes de serviço diversas, em vez de um único serviço de melhor esforço

TABELA 7.4 TRÊS TÉCNICAS EM NÍVEL DE REDE PARA DAR SUPORTE A APLICAÇÕES DE MULTIMÍDIA

Técnica	Granularidade	Garantia	Mecanismos	Complexidade	Implementação no momento
Obtendo o melhor do serviço de melhor esforço	todo o tráfego tratado igualmente	nenhuma ou flexível	suporte da camada de aplicação, CDNs, sobreposições, provisão de recurso em nível de rede	mínima	em toda a parte
Serviço diferenciado	diferentes classes de tráfego tratadas de formas diferentes	nenhuma ou flexível	marcação, regulação e programação de pacotes	média	alguma
Garantias de qualidade de serviço (QoS) por conexão	cada fluxo de origem-destino tratado de forma diferente	flexível ou rígida, uma vez admitido o fluxo	marcação, regulação e programação de pacotes; admissão e sinalização de chamadas	leve	pouca

abrangente. Com o **serviço diferenciado**, um tipo de tráfego poderia receber prioridade estrita sobre outra classe de tráfego quando os dois tipos forem enfileirados em um roteador. Por exemplo, pacotes pertencentes a uma aplicação interativa em tempo real poderiam ter prioridade sobre outros pacotes, por causa de restrições de atraso mais rigorosas. A introdução do serviço diferenciado na rede exigirá novos mecanismos para marcação de pacotes (indicando a classe de serviço de um pacote), programação de pacotes e outros. Veremos o serviço diferenciado, e novos mecanismos de rede necessários para executar esse serviço, na Seção 7.5.2.

- *Garantias de qualidade de serviço (QoS) por conexão.* Com garantias de QoS por conexão, cada instância de uma aplicação reserva explicitamente largura de banda de fim a fim e, assim, tem um desempenho de fim a fim garantido. Uma **garantia rígida** significa que a aplicação receberá sua qualidade de serviço (QoS) requisitada com certeza. Uma **garantia flexível** significa que a aplicação receberá sua qualidade de serviço requisitada com alta probabilidade. Por exemplo, se um usuário quiser fazer uma chamada VoIP do Hospedeiro A para o Hospedeiro B, a aplicação VoIP do usuário reserva largura de banda explicitamente em cada enlace ao longo de uma rota entre os dois hospedeiros. Porém, permitir que as aplicações façam reservas e exigir que a rede cumpra as reservas requer algumas grandes mudanças. Primeiro, precisamos de um protocolo que, em favor das aplicações, reserve largura de banda do enlace nos caminhos dos remetentes aos destinatários. Segundo, precisaremos de novas políticas de escalonamento nas filas do roteador, para que as reservas de largura de banda por conexão possam ser cumpridas. Por fim, para fazer uma reserva, as aplicações precisam dar à rede uma descrição do tráfego que elas pretendem enviar, e a rede precisará regular o tráfego de cada aplicação para ter certeza de que obedecerá a essa descrição. Tais mecanismos, quando combinados, exigem software novo e complexo nos hospedeiros e roteadores. Como o serviço garantido de QoS por conexão não teve implementação significativa, veremos esses mecanismos apenas rapidamente na Seção 7.5.3.

7.5.1 Dimensionando redes de melhor esforço

Fundamentalmente, a dificuldade no suporte de aplicações de multimídia surge de seus requisitos de desempenho rigorosos — pouco atraso, variação de atraso e perda de pacotes de fim a fim — e do fato de que atraso, variação do atraso e perda de pacotes ocorrem sempre que a rede fica congestionada. Uma primeira técnica para melhorar a qualidade das aplicações de multimídia — e que muitas vezes pode ser usada para resolver quase qualquer problema onde os recursos são restritos — é apenas "gastar dinheiro no problema" e, desse modo,

evitar disputa por recursos. No caso da multimídia em rede, isso significa oferecer capacidade de enlace suficiente através da rede para que nunca (ou raramente) ocorra congestionamento, além de consequentes atrasos e perdas de pacotes. Com capacidade de enlace suficiente, os pacotes poderiam correr a Internet de hoje sem atraso ou perda nas filas. Por muitos pontos de vista, essa é uma situação ideal — aplicações de multimídia funcionariam perfeitamente, os usuários estariam satisfeitos e tudo isso poderia ser obtido sem mudanças na arquitetura de melhor esforço da Internet.

A questão, logicamente, é quanta capacidade é "suficiente" para se conseguir esse paraíso, e se os custos para fornecer largura de banda "suficiente" são práticos do ponto de vista comercial para os ISPs. A questão de quanta capacidade oferecer nos enlaces de rede em determinada topologia para alcançar determinado nível de desempenho costuma ser conhecida como **provisão de largura de banda**. A questão ainda mais complicada de como projetar uma topologia de rede (onde posicionar roteadores, como interconectar roteadores com enlaces e que capacidade atribuir aos enlaces) para alcançar determinado nível de desempenho de fim a fim é um problema de projeto de redes que muitas vezes é chamado de **dimensionamento de redes**. Tanto a provisão de largura de banda quanto o dimensionamento de rede são assuntos complexos, bem além do escopo deste livro. Observamos aqui, porém, que as seguintes questões precisam ser resolvidas para que se possa prever o desempenho em nível de aplicação entre duas extremidades da rede, oferecendo assim capacidade suficiente para atender aos requisitos de desempenho de uma aplicação.

- *Modelos de demanda de tráfego entre as extremidades da rede.* Os modelos podem precisar ser especificados no nível de chamada (por exemplo, os usuários "chegando" à rede e executando aplicações) e no nível de pacote (por exemplo, pacotes sendo gerados pelas aplicações em curso). Observe que a carga de trabalho pode variar com o tempo.
- *Requisitos de desempenho bem definidos.* Por exemplo, um requisito de desempenho para dar suporte ao tráfego sensível ao atraso, como uma aplicação de multimídia interativa, poderia ser que a probabilidade de o atraso de fim a fim do pacote ser maior que um atraso máximo tolerável seja menor do que algum valor pequeno [Fraleigh, 2003].
- *Modelos para prever o desempenho de fim a fim para determinado modelo de carga de trabalho, e técnicas para encontrar uma alocação de largura de banda de custo mínimo que resulte em cumprir todos os requisitos do usuário.* Aqui, os pesquisadores estão ocupados desenvolvendo modelos de desempenho que possam quantificar o desempenho para determinada carga de trabalho, e técnicas de otimização para achar alocações de largura de banda de custo mínimo cumprindo requisitos de desempenho.

Sabendo que a Internet de melhor esforço de hoje poderia (do ponto de vista da tecnologia) dar suporte para o tráfego de multimídia em um nível de desempenho apropriado, se fosse dimensionada para fazer isso, a questão é por que a Internet de hoje não o faz. As respostas são principalmente econômicas e organizacionais. De um ponto de vista econômico, os usuários estariam dispostos a pagar a seus ISPs o suficiente para que eles instalem largura de banda suficiente para dar suporte a aplicações de multimídia por uma Internet de melhor esforço? As questões organizacionais talvez sejam ainda mais assombrosas. Observe que um caminho de fim a fim entre duas extremidades de multimídia passará pelas redes de vários ISPs. De um ponto de vista organizacional, esses ISPs estariam dispostos a cooperar (talvez compartilhando receitas) para garantir que o caminho de fim a fim seja dimensionado devidamente para dar suporte a aplicações de multimídia? Para obter uma perspectiva sobre essas questões econômicas e organizacionais, consulte Davies [2005]. Para obter uma perspectiva sobre a provisão de redes de *backbone* da camada 1 para dar suporte ao tráfego sensível a atrasos, consulte Fraleigh [2003].

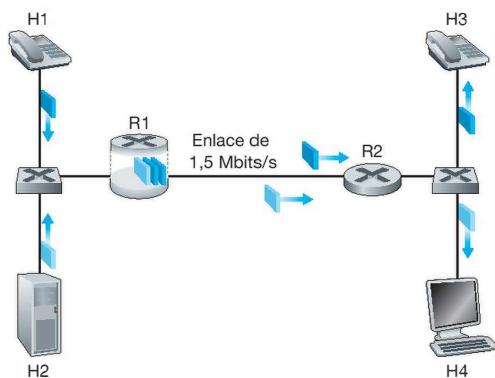
7.5.2 Fornecendo múltiplas classes de serviço

Talvez a melhoria mais simples no serviço de melhor esforço da Internet do tipo "tamanho único" atualmente seja dividir o tráfego em classes e fornecer diferentes níveis de serviço para essas classes. Por exemplo,

um ISP poderia querer oferecer uma classe de serviço mais alta para o tráfego de VoIP sensível ao atraso ou teleconferência (e cobrar mais por esse serviço!) do que para o tráfego elástico, como e-mail ou HTTP. Como alternativa, um ISP pode simplesmente querer oferecer uma qualidade de serviço mais alta para clientes que queiram pagar mais por esse serviço melhorado. Diversos ISPs de acesso residencial com fio e ISPs de acesso sem fio por celular adotaram esses níveis de serviço em camadas — com assinantes de serviço "platina" recebendo um serviço melhor do que os assinantes "ouro" ou "prata".

Já estamos acostumados com diferentes classes de serviço em nossas vidas diárias — passageiros de primeira classe em companhias aéreas recebem melhor atendimento do que os da classe executiva, que por sua vez recebem melhor atendimento do que aqueles que voam na classe econômica; VIPs recebem entrada imediata em eventos, enquanto todos os outros esperam na fila; idosos são respeitados em alguns países e recebem assentos de honra e a melhor comida em uma mesa. É importante observar que esse serviço diferencial é fornecido entre agregações de tráfego, ou seja, entre classes de tráfego, e não entre conexões individuais. Por exemplo, todos os passageiros de primeira classe são tratados da mesma forma (sem que qualquer passageiro da primeira classe receba tratamento melhor do que qualquer outro da mesma classe), assim como todos os pacotes VoIP receberiam o mesmo tratamento dentro da rede, sem depender da conexão fim a fim à qual pertencem. Conforme veremos, lidando com um pequeno número de agregações de tráfego, em vez de um número grande de conexões individuais, novos mecanismos de rede exigidos para fornecer um serviço ainda melhor podem ser mantidos relativamente simples.

Os primeiros projetistas da Internet claramente tinham essa noção de múltiplas classes de serviço em mente. Relembre do campo tipo de serviço (ToS) no cabeçalho IPv4 na Figura 4.13. IEN123 [ISI 1979] descreve o campo ToS, também presente em um ancestral do datagrama IPv4, como segue:

> *O tipo de serviço [campo] fornece uma indicação dos parâmetros abstratos da qualidade de serviço desejada. Esses parâmetros devem ser usados para guiar a escolha de parâmetros através de serviço quando um datagrama estiver sendo transmitido através de uma rede específica. Muitas redes oferecem serviços prioritários, os quais de alguma forma reservam mais atenção ao tráfego de prioridade alta do que aos outros.*

Até mesmo há três décadas, a visão de fornecer diferentes níveis de serviço a diferentes níveis de tráfego estava evidente. No entanto, levou um longo período para que pudéssemos perceber essa visão.

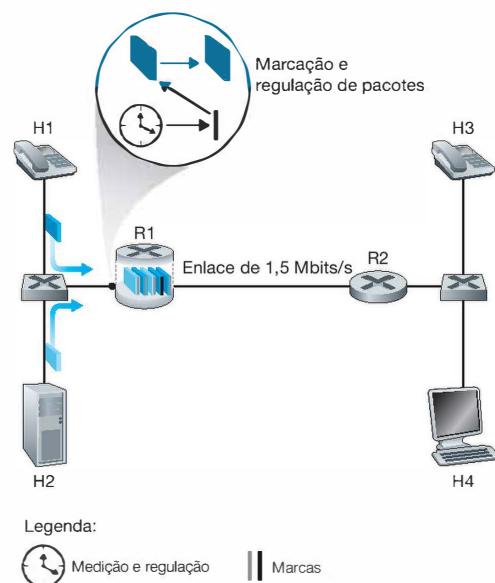
Cenários motivadores

Vamos começar nossa discussão sobre mecanismos de rede oferecendo várias classes de serviço com alguns cenários motivadores.

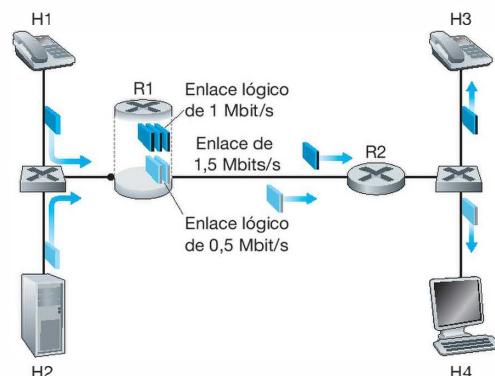
A Figura 7.14 mostra um cenário simples de rede em que dois fluxos de pacotes de aplicação se originam nos hospedeiros H1 e H2 em uma LAN e são destinados aos hospedeiros H3 e H4 em outra LAN. Os roteadores nas duas LANs são ligados por um enlace de 1,5 Mbits/s. Vamos admitir que as velocidades das LANs sejam bem mais altas do que 1,5 Mbits/s e vamos focalizar a fila de saída do roteador R1; é aqui que ocorrerão atraso e perda de pacotes se a taxa agregada de envio de H1 e H2 exceder 1,5 Mbits/s. Vamos supor ainda que uma aplicação de áudio de 1 Mbit/s (por exemplo, uma chamada de áudio com qualidade de CD) compartilhe o enlace de 1,5 Mbits/s entre R1 e R2 com uma aplicação de navegação Web HTTP que está baixando uma página Web de H2 para H4.

Na Internet de melhor esforço, os pacotes de áudio e HTTP são misturados na fila de saída de R1 e (em geral) transmitidos na ordem primeiro a entrar/primeiro a sair (*first-in-first-out* — FIFO). Nesse cenário, uma rajada de pacotes da origem HTTP poderia lotar a fila, fazendo pacotes IP de áudio sofrerem atraso excessivo ou perda, por causa do estouro do buffer em R1. Como resolveríamos esse problema potencial? Dado que a aplicação de navegação Web HTTP não tem limitação de tempo, poderíamos, por intuição, dar prioridade estrita aos pacotes de áudio em R1. Em uma disciplina estrita de escalonamento por prioridade, um pacote IP de áudio no buffer de saída de R1 seria sempre transmitido antes de qualquer pacote HTTP nesse mesmo buffer. O enlace de

FIGURA 7.14 COMPETINDO APLICAÇÕES DE ÁUDIO E HTTP

R1 a R2 pareceria um enlace dedicado de 1,5 Mbits/s para o tráfego de áudio e o HTTP usaria o enlace de R1 a R2 somente quando não houvesse nenhum tráfego de áudio na fila. Para R1 distinguir os pacotes de áudio dos HTTP em sua fila, cada pacote deve ser marcado como pertencente a uma das duas classes de tráfego. Lembre-se de que esse era o objetivo original do campo de tipo de serviço (*type-of-service* — ToS) do IPv4. Por mais óbvio que pareça, esse é, então, nosso primeiro princípio dos mecanismos necessários para oferecer múltiplas classes de tráfego:

> **Princípio 1:** A **marcação de pacotes** permite que um roteador faça a distinção de pacotes pertencentes a diferentes classes de tráfego.

Observe que, embora nosso exemplo considere um fluxo concorrente de multimídia e elástico, o mesmo princípio se aplica ao caso em que classes de serviço platina, ouro e prata são implementadas — um mecanismo de marcação de pacotes ainda é necessário para indicar a classe de serviço à qual um pacote pertence.

Suponha agora que o roteador saiba que deve dar prioridade a pacotes da aplicação de áudio de 1 Mbit/s. Como a velocidade de saída do enlace é de 1,5 Mbits/s, mesmo que os pacotes HTTP recebam prioridade mais baixa, ainda assim receberão, na média, um serviço de transmissão de 0,5 Mbit/s. Mas o que acontece se a aplicação de áudio começar a enviar pacotes a uma taxa de 1,5 Mbits/s ou mais alta (seja com má intenção, seja devido a um erro de aplicação)? Nesse caso, os pacotes HTTP morrerão por inanição, isto é, não receberão nenhum serviço no enlace de R1 a R2. Problemas semelhantes ocorreriam se várias aplicações (por exemplo, várias chamadas de áudio), todas com a mesma classe de serviço, estivessem compartilhando a largura de banda de um enlace; elas também poderiam coletivamente arruinar a sessão HTTP. Idealmente, o que se quer é certo grau de isolamento entre classes de tráfego, para proteger uma classe da outra. Essa proteção poderia ser implementada em diferentes locais da rede — em todo e qualquer roteador, na primeira entrada na rede ou nos limites da rede entre domínios. Logo, este é o nosso segundo princípio:

> **Princípio 2:** É desejável fornecer algum grau de **isolamento de tráfego** entre as classes, para que uma classe não seja afetada adversamente por outra classe de comportamento inadequado.

Examinaremos diversos mecanismos específicos que fornecem isolamento entre classes de tráfego. Observamos aqui que podem ser adotadas duas abordagens amplas. Com a primeira, é possível realizar a **regulação de tráfego**, como mostra a Figura 7.15. Se uma classe ou fluxo de tráfego deve se ajustar a certos critérios (por exemplo, o fluxo de áudio não deve exceder uma taxa de pico de 1 Mbit/s), então um mecanismo de regulação pode ser introduzido para assegurar que esse critério seja, de fato, observado. Se a aplicação regulada se comportar mal, o mecanismo de regulação executará alguma ação (por exemplo, descartará ou atrasará pacotes que estão violando o critério), de modo que o tráfego que está entrando na rede obedeça aos critérios. O mecanismo do tipo "balde furado" (*leaky bucket*), que examinaremos mais adiante, talvez seja o mecanismo de regulação mais utilizado.

FIGURA 7.15 REGULAÇÃO (E MARCAÇÃO) DAS CLASSES DE TRÁFEGO DE ÁUDIO E HTTP

Marcação e regulação de pacotes
H1
H3
R1
Enlace de 1,5 Mbits/s
R2
H2
H4
Legenda:
Medição e regulação
Marcas

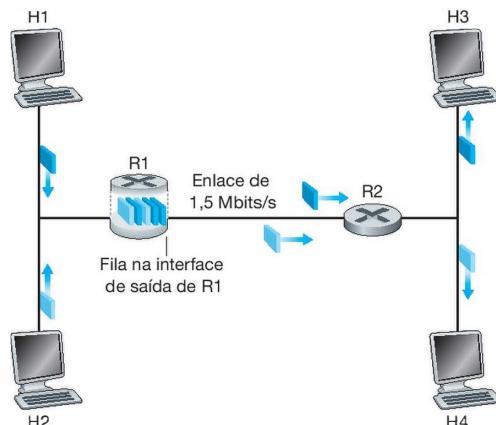
Na Figura 7.15, o mecanismo de classificação e marcação de pacotes (Princípio 1) e o mecanismo de regulação (Princípio 2) estão localizados juntos na borda da rede, seja no sistema final, seja em um roteador de borda.

Uma abordagem alternativa para prover isolamento entre classes de tráfego é deixar para o mecanismo de programação de pacotes na camada de enlace a tarefa de alocar explicitamente uma parte fixa da largura de banda de enlace a cada classe. Por exemplo, a classe de áudio poderia ser alocada em R1 a 1 Mbit/s, e a classe HTTP, 0,5 Mbit/s. Nesse caso, os fluxos de áudio e HTTP perceberiam um enlace lógico com capacidade de 1 Mbit/s e 0,5 Mbit/s, respectivamente, conforme mostra a Figura 7.16. Com imposição estrita de alocação de largura de banda na camada de enlace, um fluxo pode usar apenas a quantidade de largura de banda que lhe foi alocada; em particular, ele não pode utilizar largura de banda que não está sendo usada no momento pelas outras aplicações.

FIGURA 7.16 ISOLAMENTO LÓGICO DAS CLASSES DE TRÁFEGO DE ÁUDIO E HTTP

Por exemplo, se o fluxo de áudio silenciasse (se o interlocutor fizesse uma pausa e não gerasse nenhum pacote de áudio, por exemplo), ainda assim o fluxo HTTP não conseguiria transmitir mais do que 0,5 Mbit/s pelo enlace de R1 a R2, mesmo que a alocação de largura de banda de 1 Mbit/s para o fluxo de áudio não estivesse sendo utilizada naquele momento. Como a largura de banda é um recurso do tipo "use ou perca", não há motivo para impedir que o tráfego HTTP use a largura de banda não usada pelo tráfego de áudio. É desejável usar a largura de banda da maneira mais eficiente possível, nunca a desperdiçando quando puder ser usada para outra finalidade. Essa consideração origina nosso terceiro princípio:

> **Princípio 3:** Ao fornecer isolamento de classes ou fluxos, é desejável que se usem os recursos (por exemplo, largura de banda de enlace e buffers) da maneira mais eficiente possível.

Mecanismos de escalonamento

Lembre-se de que dissemos na Seção 1.3 e na Seção 4.3 que pacotes pertencentes a vários fluxos de rede são multiplexados e enfileirados para transmissão nos buffers de saída associados a um enlace. O modo como os pacotes enfileirados são selecionados para transmissão pelo enlace é conhecido como **disciplina de escalonamento do enlace**. Vamos agora examinar mais detalhadamente algumas das mais importantes disciplinas de escalonamento.

Primeiro a entrar/primeiro a sair (FIFO)

A Figura 7.17 mostra a representação do modelo de enfileiramento para a disciplina de escalonamento de enlace primeiro a entrar/primeiro a sair (FIFO). Pacotes que chegam à fila de saída do enlace esperam pela transmissão se, naquele momento, o enlace estiver ocupado com a transmissão de outro pacote. Se não houver espaço suficiente de buffer para guardar o pacote que chega, a **política de descarte de pacotes** da fila então determinará se o pacote será descartado (perdido) ou se outros serão retirados da fila para dar espaço ao que está chegando. Em nossa discussão a seguir, vamos ignorar o descarte de pacotes. Quando um pacote é transmitido integralmente pelo enlace de saída (isto é, recebe serviço), ele é retirado da fila.

A disciplina de escalonamento FIFO — também conhecida como FCFS (*first-come-first-served* — primeiro a chegar/primeiro a ser atendido) — seleciona pacotes para transmissão pelo enlace na mesma ordem em que eles chegaram à fila de saída do enlace. Todos estamos familiarizados com as filas FIFO em pontos de ônibus (em

FIGURA 7.17 REPRESENTAÇÃO DO ENFILEIRAMENTO FIFO

particular na Inglaterra, onde parece que as filas atingiram a perfeição) ou em agências bancárias e outros órgãos, onde os clientes que chegam se juntam ao final da fila de espera, permanecem na ordem e então são atendidos quando atingem o início da fila.

A Figura 7.18 mostra a fila FIFO em operação. As chegadas de pacotes são indicadas por setas numeradas acima da linha de tempo superior; os números indicam a ordem em que os pacotes chegaram. As saídas de pacotes individuais são mostradas abaixo da linha de tempo inferior. O tempo que um pacote passa no atendimento (sendo transmitido) é indicado pelo retângulo sombreado entre as duas linhas de tempo. Por causa da disciplina FIFO, os pacotes saem na mesma ordem em que chegaram. Note que, após a saída do pacote 4, o enlace permanece ocioso (uma vez que os pacotes 1 a 4 já foram transmitidos e retirados da fila) até a chegada do pacote 5.

Enfileiramento prioritário

Pela regra do enfileiramento prioritário, pacotes que chegam ao enlace de saída são classificados em classes de prioridade na fila de saída, como mostra a Figura 7.19. Como discutimos na seção anterior, a classe de prioridade de um pacote pode depender de uma marca explícita que ele carrega em seu cabeçalho (por exemplo, o valor dos bits de ToS em um pacote IPv4), seu endereço IP de origem ou destino, seu número de porta de destino ou outro critério. Cada classe de prioridade tem em geral sua própria fila. Ao escolher um pacote para transmitir, a disciplina de enfileiramento prioritário transmitirá um pacote da classe de prioridade mais alta cuja fila não esteja vazia (isto é, tenha pacotes esperando transmissão). A escolha entre pacotes *da mesma classe de prioridade* é feita, normalmente, pelo método FIFO.

A Figura 7.20 ilustra a operação de uma fila prioritária com duas classes de prioridade. Os pacotes 1, 3 e 4 pertencem à classe de alta prioridade e os pacotes 2 e 5, à classe de baixa prioridade. O pacote 1 chega e, encontrando o enlace vazio, inicia a transmissão. Durante a transmissão do pacote 1, os pacotes 2 e 3 chegam e são colocados nas filas de baixa e de alta prioridade, respectivamente. Após a transmissão do pacote 1, o pacote 3

FIGURA 7.18 A FILA FIFO EM OPERAÇÃO

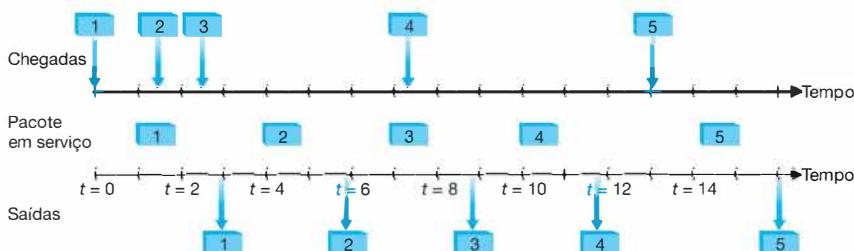



FIGURA 7.19 MODELO DE ENFILEIRAMENTO PRIORITÁRIO

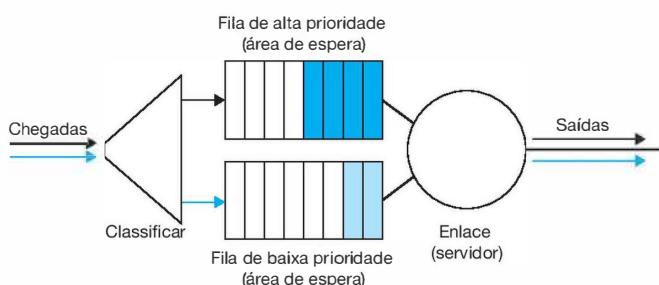
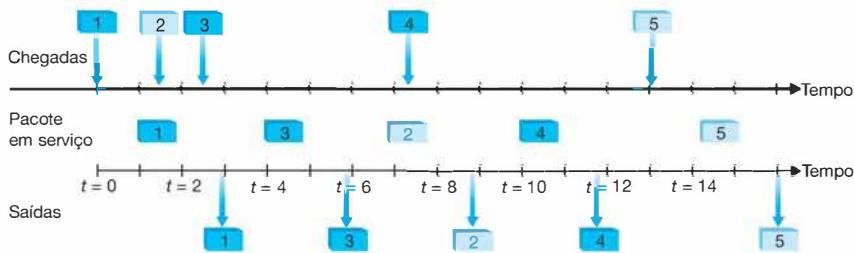
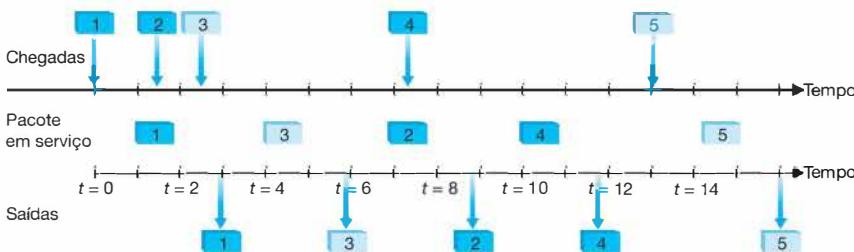



FIGURA 7.20 OPERAÇÃO DO ENFILEIRAMENTO PRIORITÁRIO

(um pacote de alta prioridade) é selecionado para transmissão, passando à frente do pacote 2 (que, mesmo tendo chegado primeiro, é de baixa prioridade). Ao término da transmissão do pacote 3, começa a transmissão do pacote 2. O pacote 4 (de alta prioridade) chega durante a transmissão do pacote 2 (de baixa prioridade). Em uma disciplina de enfileiramento prioritário não preemptiva, a transmissão de um pacote não será interrompida se já tiver começado. Nesse caso, o pacote 4 entra na fila para transmissão e começa a ser transmitido após a conclusão da transmissão do pacote 2.

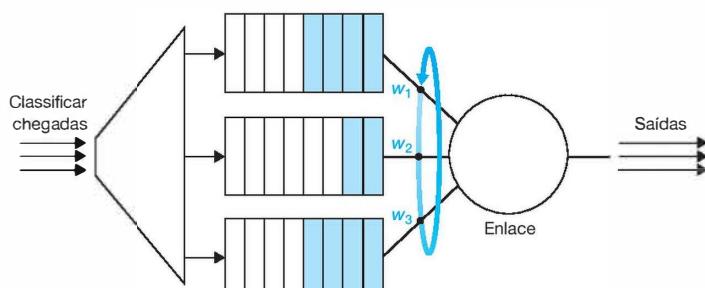
Varredura cíclica e WQF (enfileiramento justo ponderado)

Na **disciplina de enfileiramento por varredura cíclica**, pacotes são classificados do mesmo modo que no enfileiramento prioritário. Contudo, em vez de haver uma prioridade estrita de serviço entre as classes, um escalonador de varredura cíclica alterna serviços entre elas. Na forma mais simples desse escalonamento, um pacote de classe 1 é transmitido, seguido por um pacote de classe 2, seguido por um pacote de classe 1, seguido por um pacote de classe 2 e assim por diante. Uma disciplina de enfileiramento de conservação de trabalho nunca permitirá que o enlace fique ocioso enquanto houver pacotes (de qualquer classe) enfileirados para transmissão. Uma **disciplina de varredura cíclica de conservação de trabalho** que procura um pacote de determinada classe, mas não encontra nenhum, verifica imediatamente a classe seguinte na sequência da varredura cíclica.

A Figura 7.21 ilustra a operação de uma fila de duas classes por varredura cíclica. Nesse exemplo, os pacotes 1, 2 e 4 pertencem à classe 1 e os pacotes 3 e 5, à classe 2. O pacote 1 inicia a transmissão imediatamente após sua chegada à fila de saída. Os pacotes 2 e 3 chegam durante a transmissão do pacote 1 e, assim, entram na fila. Após a transmissão do pacote 1, o escalonador de enlace procura um pacote de classe 2 e, então, transmite o pacote 3. Após a transmissão do pacote 3, o escalonador procura um pacote de classe 1 e, então, transmite o pacote 2. Após a transmissão do pacote 2, o pacote 4 é o único na fila; assim, ele é transmitido imediatamente após o pacote 2.

Uma abstração generalizada do enfileiramento por varredura cíclica que encontrou considerável utilização nas arquiteturas com QoS é a denominada disciplina de **enfileiramento justo ponderado** (*weighted fair*

FIGURA 7.21 OPERAÇÃO DE ENFILEIRAMENTO DE DUAS CLASSES POR VARREDURA CÍCLICA

queuing — WFQ) [Demers, 1990; Parekh, 1993]. A WFQ é ilustrada na Figura 7.22. Os pacotes que chegam são classificados e enfileirados por classe em suas áreas de espera apropriadas. Como acontece no escalonamento por varredura cíclica, um programador WFQ atende às classes de modo cíclico — atende primeiro à classe 1, depois à classe 2 e, em seguida, à classe 3; e então (supondo que haja três classes) repete o esquema de serviço. A WFQ também é uma disciplina de enfileiramento de conservação de trabalho; assim, ao encontrar uma fila de classe vazia, ela imediatamente passa para a classe seguinte na sequência de atendimento.

A WFQ é diferente da varredura cíclica, pois cada classe pode receber uma quantidade de serviço *diferenciado* a qualquer intervalo de tempo. Em particular, a cada classe i é atribuído um peso w_i. A WFQ garante que, em qualquer intervalo de tempo durante o qual houver pacotes da classe i para transmitir, a classe i receberá uma fração de serviço igual a $w_i/(\sum w_j)$, onde o denominador é a soma de todas as classes que também têm pacotes enfileirados para transmissão. No pior caso, mesmo que todas as classes tenham pacotes na fila, a classe i ainda terá garantido o recebimento de uma fração $w_i / (\sum w_j)$ da largura de banda. Assim, para um enlace com taxa de transmissão R, a classe i sempre conseguirá uma vazão de, no mínimo, $R \cdot w_i/(\sum w_j)$. Descrevemos a WFQ em condições ideais, pois não consideramos o fato de que os pacotes são unidades discretas de dados e que a transmissão de um pacote não será interrompida para dar início à transmissão de outro; Demers [1990] e Parekh [1993] discutem essa questão do empacotamento. Como veremos nas seções seguintes, a WFQ tem um papel fundamental nas arquiteturas com QoS. Ela também está disponível nos roteadores fabricados atualmente [Cisco QoS, 2012].

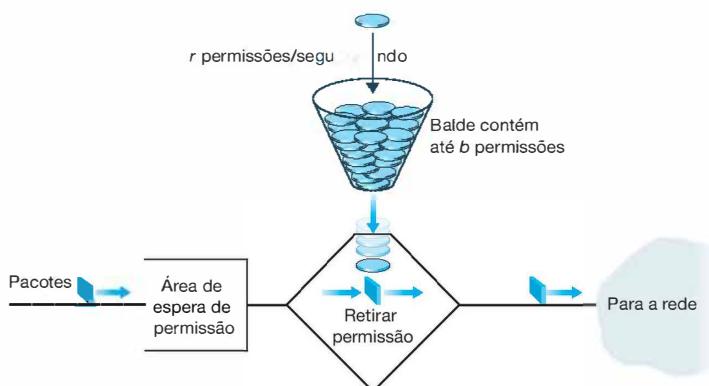
Regulação: o "balde furado"

Um dos nossos princípios foi que a regulação da taxa com a qual é permitido que um fluxo (vamos supor que uma unidade de regulação é um fluxo na nossa discussão a seguir) injete pacotes na rede, é um importante mecanismo de QoS. Mas quais características da taxa de pacotes de um fluxo devem ser reguladas? Podemos identificar três critérios importantes de regulação diferentes entre si pela escala de tempo durante a qual o pacote é regulado:

- *Taxa média*. A rede pode desejar limitar a taxa média durante um período de tempo (pacotes por intervalo de tempo) com a qual os pacotes de um fluxo podem ser enviados para a rede. Uma questão crucial aqui é o intervalo de tempo durante o qual a taxa média será regulada. Um fluxo cuja taxa média está limitada a 100 pacotes por segundo é mais restringido do que uma origem limitada a 6.000 pacotes por minuto, mesmo que ambos tenham a mesma taxa média durante um intervalo de tempo longo o suficiente. Por exemplo, a última limitação permitiria que um fluxo enviasse 1.000 pacotes em dado intervalo de tempo de um segundo de duração, enquanto a limitação anterior desautorizaria esse comportamento de envio.
- *Taxa de pico*. Enquanto a limitação da taxa média restringe a quantidade de tráfego que pode ser enviada para a rede durante um período de tempo relativamente longo, uma limitação de taxa de pico restringe o

FIGURA 7.22 ENFILEIRAMENTO JUSTO PONDERADO (WFQ)

número máximo de pacotes que podem ser enviados durante um período de tempo mais curto. Usando o mesmo exemplo anterior, a rede pode regular um fluxo a uma taxa média de 6.000 pacotes por minuto e, ao mesmo tempo, limitar a taxa de pico do fluxo a 1.500 pacotes por segundo.

- *Tamanho da rajada*. A rede também pode limitar o número máximo de pacotes (a "rajada" de pacotes) que podem ser enviados para dentro dela durante um intervalo curtíssimo de tempo. No limite, à medida que o comprimento do intervalo se aproxima de zero, o tamanho da rajada limita o número de pacotes que podem ser enviados instantaneamente para a rede. Mesmo que seja fisicamente impossível enviar vários pacotes para a rede instantaneamente (afinal, cada enlace tem uma taxa de transmissão física que não pode ser ultrapassada!), a abstração de um tamanho máximo de rajada é útil.

O mecanismo de balde furado (*leaky bucket*) é uma abstração que pode ser usada para caracterizar esses limites da regulação. Como mostra a Figura 7.23, um balde furado é um balde que pode conter até b permissões. As permissões são adicionadas da seguinte forma. Novas permissões, que potencialmente seriam adicionadas ao balde, estão sempre sendo geradas a uma taxa de r permissões por segundo. (Para facilitar, nesse caso admitimos que a unidade de tempo seja 1 s.) Se o balde estiver cheio com menos de b permissões quando for gerada uma permissão, esta será adicionada ao balde; caso contrário, será ignorada e o balde permanecerá cheio com as b permissões.

Vamos considerar agora como o balde furado pode ser usado para regular um fluxo de pacotes. Suponha que, antes de um pacote ser transmitido para a rede, primeiro ele tenha de retirar uma permissão de dentro do balde de permissões. Se este estiver vazio, o pacote terá de esperar por uma permissão. (Uma alternativa é o pacote ser descartado, mas não vamos considerar essa opção aqui.) Vamos considerar agora como esse comportamento regula o fluxo de tráfego. Como pode haver no máximo b permissões no balde, o tamanho máximo da rajada para um fluxo regulado pela técnica do balde furado é b pacotes. Além disso, como a taxa de geração de permissões é r, o número máximo de pacotes que pode entrar na rede para qualquer intervalo de tempo t é $rt + b$. Assim, a taxa de geração de permissões, r, serve para limitar a taxa média a longo prazo com a qual pacotes podem entrar na rede. Também é possível usar baldes furados (especificamente, dois baldes em série) para regular a taxa de pico de um fluxo e a taxa média a longo prazo; veja os exercícios ao final deste capítulo.

Balde furado + enfileiramento justo ponderado = máximo atraso provável em uma fila

Vamos fechar nossa discussão sobre escalonamento e regulação mostrando como os dois podem ser combinados para oferecer um limite sobre o atraso na fila de um roteador. Vamos considerar o enlace de saída de um roteador que multiplexa n fluxos, cada um regulado por um balde furado com parâmetros b_i e r_i, $i = 1, \ldots n$,

FIGURA 7.23 O REGULADOR DO BALDE FURADO

usando o escalonamento WFQ. Usamos aqui o termo *fluxo* de modo um pouco impreciso para denominar os conjuntos de pacotes que não são distinguidos uns dos outros pelo escalonador. Na prática, um fluxo pode conter o tráfego de uma única conexão fim a fim ou um conjunto de muitas conexões desse tipo; veja a Figura 7.24.

Lembre-se de que mencionamos, em nossa discussão sobre a WFQ, que cada fluxo i tem a garantia de receber uma parcela da largura de banda do enlace igual a, no mínimo, $R \cdot w_i/(\sum w_j)$, sendo R a taxa de transmissão do enlace em pacotes por segundo. Qual é, então, o atraso máximo que sofrerá um pacote enquanto espera ser atendido na WFQ (isto é, após ter passado pelo balde furado)? Vamos considerar o fluxo 1. Suponha que o balde de permissões do fluxo 1 esteja de início cheio. Uma rajada de b_1 pacotes então chega ao regulador de balde furado para o fluxo 1. Os pacotes retiram todas as permissões do balde (sem esperar) e, em seguida, juntam-se à área de espera WFQ para o fluxo 1. Como esses b_1 pacotes são atendidos a uma taxa de, no mínimo, $R \cdot w_i/(\sum w_j)$ pacotes por segundo, o último deles sofrerá um atraso máximo, $d_{máx}$, até que sua transmissão seja concluída, onde

$$d_{max} = \frac{b_1}{R \cdot w_i / \sum w_j}$$

A razão dessa fórmula é que, se houver b_1 pacotes na fila e eles estiverem sendo atendidos (retirados) na fila a uma taxa de, no mínimo, $R \cdot w_i/(\sum w_j)$ por segundo, então a quantidade de tempo até que o último bit do último pacote seja transmitido não poderá ser maior do que $b_1/(R \cdot w_i/(\sum w_j))$. Um exercício ao final deste capítulo solicita que você demonstre que, se $r_1 < R \cdot w_i/(\sum w_j)$, então d_{max} é o atraso máximo que qualquer pacote do fluxo 1 pode sofrer na fila WFQ.

7.5.3 Diffserv

Tendo visto a motivação, os princípios e os mecanismos específicos para oferecer múltiplas classes de serviço, vamos encerrar nosso estudo das técnicas para oferecer múltiplas classes de serviços com um exemplo — a arquitetura Diffserv [RFC 2475; Kilkki, 1999]. Diffserv oferece diferenciação de serviço — isto é, a capacidade de lidar com as diversas classes de tráfego de diferentes modos na Internet, de modo escalável. A necessidade da escalabilidade vem do fato que centenas de milhares de fluxos simultâneos de tráfego de origem-destino estão presentes em um roteador *backbone*. Veremos de modo breve que essa necessidade é satisfeita ao colocarmos uma funcionalidade dentro do núcleo da rede, com operações de controle mais complexas sendo implementadas na borda da rede.

Vamos começar com a rede simples mostrada na Figura 7.25. Descreveremos um possível uso da arquitetura Diffserv; outras variações são possíveis, como descreve o RFC 2475. A arquitetura Diffserv consiste em dois conjuntos de elementos funcionais:

FIGURA 7.24 ***N* FLUXOS MULTIPLEXADOS COM BALDE FURADO COM ESCALONAMENTO WFQ**

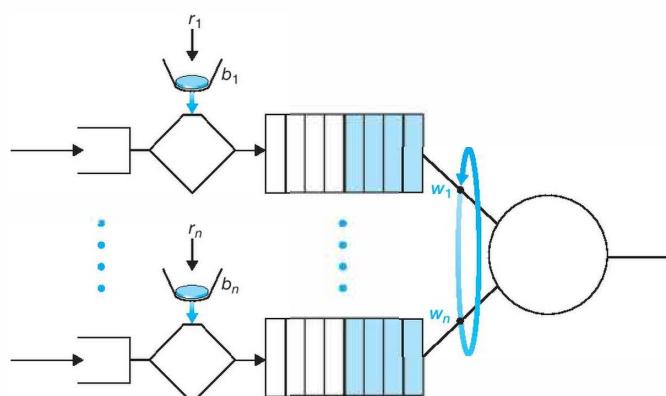
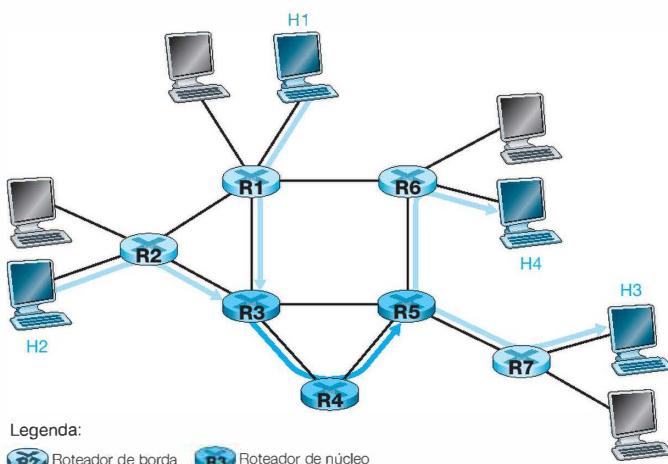
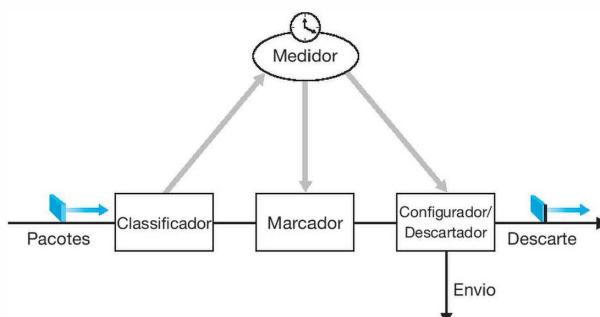



FIGURA 7.25 EXEMPLO SIMPLES DE REDE DIFFSERV

- *Funções de borda: classificação de pacotes e condicionamento de tráfego*. Na borda de entrada da rede (isto é, ou em um hospedeiro habilitado para Diffserv que gera o tráfego, ou no primeiro roteador habilitado para Diffserv pelo qual o tráfego passa), os pacotes que chegam são marcados. Mais especificamente, o campo Differentiated Service (DS) do cabeçalho do pacote IPv4 ou IPv6 é definido para algum valor [RFC 3260]. A definição do campo DS teve por finalidade substituir as definições mais antigas do campo de tipo de serviço do IPv4 e os campos de classe de tráfego do IPv6 que discutimos no Capítulo 4. Por exemplo, na Figura 7.25, os pacotes que estão sendo enviados de H1 para H3 poderiam ser marcados em R1, ao passo que os pacotes enviados de H2 para H4 poderiam ser marcados em R2. A marca que um pacote recebe identifica a classe de tráfego à qual ele pertence. Assim, diferentes classes de tráfego receberão serviços diferenciados dentro do núcleo da rede.
- *Função central: envio*. Quando um pacote marcado com DS chega a um roteador habilitado para Diffserv, ele é repassado até seu próximo salto de acordo com o comportamento por salto (PHB) associado à classe do pacote. O comportamento por salto influencia a maneira pela qual os buffers e a largura de banda de enlace de um roteador são compartilhados entre as classes de tráfego concorrentes. Um dogma crucial da arquitetura Diffserv é que o comportamento por salto do roteador se baseará somente nas marcas dos pacotes, isto é, na classe de tráfego a que o pacote pertence. Assim, se os pacotes que estão sendo enviados de H1 para H3 na Figura 7.25 receberem a mesma marca dos que estão sendo enviados de H2 para H4, os roteadores da rede tratarão esses pacotes como um agregado, sem distinguir se eles se originam de H1 ou H2. Por exemplo, R3 não faria nenhuma distinção entre pacotes de H1 e H2 ao transmiti-los a R4. Portanto, a arquitetura de serviço diferenciado evita a necessidade de manter o estado do roteador para pares origem-destino individuais — uma consideração importante para atender aos requisitos de escalabilidade do Diffserv.

Uma analogia poderia ser útil neste ponto. Em muitos eventos sociais de grande escala (por exemplo, uma recepção com muitos convidados, uma boate ou discoteca, um concerto ou uma partida de futebol), as pessoas que participarão do evento adquirem algum tipo de entrada. Há ingressos VIP para pessoas muito importantes; há bilhetes para maiores de 18 anos (por exemplo, para eventos em que serão servidas bebidas alcoólicas), há credenciais que dão direito a visitar o camarim dos artistas; há ingressos especiais para repórteres e imprensa em geral; há até mesmo um ingresso comum para o cidadão comum. Esses ingressos são, em geral, adquiridos em bilheterias, isto é, na borda do evento. E é aqui, na borda, que são realizadas as operações que requerem intensa

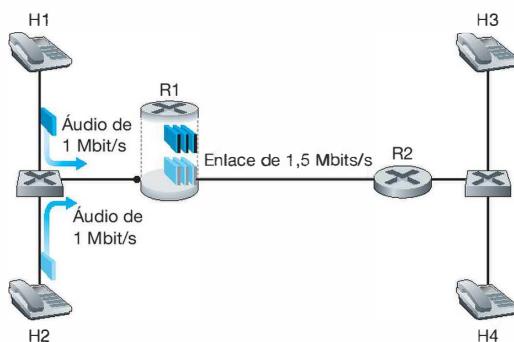
atividade computacional, como pagar pelo ingresso, verificar o tipo adequado de ingresso, comparar o ingresso com um documento de identidade. Além disso, pode haver um limite para o número de pessoas de um tipo que poderão entrar no evento. Se houver esse limite, as pessoas talvez tenham de esperar antes de entrar. Uma vez lá dentro, o tipo de ingresso determina o tipo de serviço diferenciado recebido nos diversos locais do evento — um VIP recebe bebidas gratuitas, mesa melhor, refeição gratuita, dispõe de salas exclusivas e recebe serviço especial, ao passo que uma pessoa comum não pode entrar em certas áreas, paga sua bebida e recebe apenas o serviço básico. Em ambos os casos, o serviço recebido no evento depende apenas do tipo de ingresso da pessoa. Além disso, todas as pessoas pertencentes à mesma classe recebem tratamento igual.

A Figura 7.26 apresenta uma visão lógica da função de classificação e marcação no roteador de borda. Os pacotes que chegam ao roteador de borda são primeiro classificados. O classificador seleciona os pacotes com base nos valores de um ou mais campos de cabeçalho de pacote (por exemplo, endereço de origem, endereço de destino, porta de origem, porta de destino e ID de protocolo) e dirige o pacote à função de marcação apropriada. Como já dissemos, a marca de um pacote é carregada dentro do campo DS no cabeçalho de pacote.

Em alguns casos, um usuário final pode ter concordado em limitar sua taxa de envio de pacotes conforme um **perfil de tráfego** declarado. Esse perfil poderia conter um limite à taxa de pico, bem como às rajadas do fluxo de pacotes, como já antes quando tratamos do mecanismo de balde furado. Enquanto o usuário enviar pacotes para a rede de um modo que esteja de acordo com o perfil de tráfego negociado, os pacotes receberão sua marca de prioridade e serão transmitidos ao longo de sua rota até o destino. Por outro lado, se o perfil de tráfego for violado, pacotes que estão fora do perfil poderão ser marcados de modo diferente, ajustados (por exemplo, atrasados de modo que seja observada a limitação da taxa máxima) ou descartados na borda da rede. O papel da **função de medição** mostrada na Figura 7.26 é comparar o fluxo de entrada de pacotes com o perfil de tráfego negociado. A decisão de imediatamente remarcar, repassar, atrasar ou descartar um pacote é uma questão de política determinada pelo administrador da rede e *não* está especificada na arquitetura Diffserv.

Até aqui, examinamos as funções de marcação e regulação da arquitetura Diffserv. O segundo componente fundamental dessa arquitetura envolve o comportamento por salto (PHB — *per-hop behavior*) realizado pelos roteadores habilitados para Diffserv. O PHB é definido de maneira cuidadosa, se bem que um tanto enigmática, como "uma descrição do comportamento de repasse de um nó Diffserv, que possa ser observado externamente, aplicado a um comportamento agregado Diffserv em particular" [RFC 2475]. Explorando essa definição mais a fundo, podemos ver diversas considerações importantes nela embutidas:

- Um PHB pode resultar no recebimento de diferentes desempenhos por diferentes classes de tráfego (isto é, comportamentos de repasse diferentes que possam ser observados externamente).
- Embora um PHB defina diferenças de desempenho (comportamento) entre classes, ele não determina nenhum mecanismo específico para conseguir esses comportamentos. Desde que os critérios de desempenho observados externamente sejam cumpridos, quaisquer mecanismos de execução e quaisquer

FIGURA 7.26 EXEMPLO DE UMA REDE DIFFSERV SIMPLES

políticas de alocação de buffer/largura de banda podem ser usados. Por exemplo, um PHB não exigiria que fosse utilizada uma disciplina de enfileiramento de pacotes em particular (por exemplo, um enfileiramento prioritário *versus* um enfileiramento WFQ *versus* um enfileiramento FCFS) para atingir determinado comportamento. O PHB é o fim para o qual os mecanismos de alocação e implementação de recursos são os meios.
- As diferenças de comportamento devem ser observáveis e, por conseguinte, mensuráveis.

Foram definidos dois PHBs: um PHB de repasse acelerado (*expedited forwarding* — EF) [RFC 3246] e um PHB de repasse assegurado (*assured forwarding* — AF) [RFC 2597]. O PHB de **repasse acelerado** especifica que a taxa de partida de uma classe de tráfego de um roteador deve ser igual ou maior do que uma taxa configurada. O PHB de **repasse assegurado** divide o tráfego em quatro classes e garante, a cada classe AF, o fornecimento de alguma quantidade mínima de largura de banda e de buffer.

Vamos encerrar nossa discussão do Diffserv com algumas observações com relação ao seu modelo de serviço. Primeiro, admitimos que ele é disponibilizado dentro de um único domínio administrativo. O caso mais típico é o do serviço fim a fim, que tem de ser configurado para vários ISPs localizados entre os sistemas finais comunicantes. Para prover serviço Diffserv fim a fim, todas os ISPs entre o sistema final devem não apenas prover esse serviço, mas, acima de tudo, cooperar e fazer acordos para oferecer aos clientes finais um serviço verdadeiramente fim a fim. Sem esse tipo de cooperação, ISPs que vendem serviço Diffserv diretamente a clientes terão sempre de se justificar perante eles dizendo: "Sim, sabemos que você pagou um extra, mas não temos um acordo de serviço com o ISPs que descartou e adiou o seu tráfego. Pedimos desculpas pelas muitas lacunas na sua chamada VoIP!". Segundo, se o Diffserv estivesse mesmo disponível e a rede trabalhasse apenas sob carga moderada, na maior parte do tempo não se perceberia nenhuma diferença entre um serviço de melhor esforço e um serviço Diffserv. Na verdade, hoje, o atraso fim a fim em geral é dominado pelas taxas de acesso e saltos de roteadores e não por atrasos de fila em roteadores. Imagine o infeliz cliente do Diffserv, que pagou a mais por um serviço diferenciado, mas descobre que o serviço de melhor esforço oferecido por outros quase sempre apresenta o mesmo desempenho!

7.5.4 Garantias de QoS por conexão: reserva de recurso e admissão de chamada

Na seção anterior, vimos que a marcação de pacotes e regulação, isolamento de tráfego, e escalonamento em nível de enlace podem prover uma classe de serviço com melhor desempenho do que outra. Em certas disciplinas de escalonamento, como escalonamento de prioridade, as classes inferiores de tráfego são basicamente "invisíveis" à classe de tráfego com a prioridade mais alta. Com dimensionamento adequado da rede, a classe de serviço mais alta pode, de fato, atingir taxas de perda de pacote e atraso baixíssimas — basicamente um desempenho semelhante ao circuito. Mas a rede pode *garantir* que um fluxo contínuo em uma classe de tráfego de alta prioridade continuará a receber tal atendimento pela duração do fluxo usando apenas os mecanismos que descrevemos até agora? Não pode. Nesta seção, veremos por que mecanismos e protocolos de rede adicionais são ainda necessários quando uma garantia de serviço fixa é oferecida a conexões individuais.

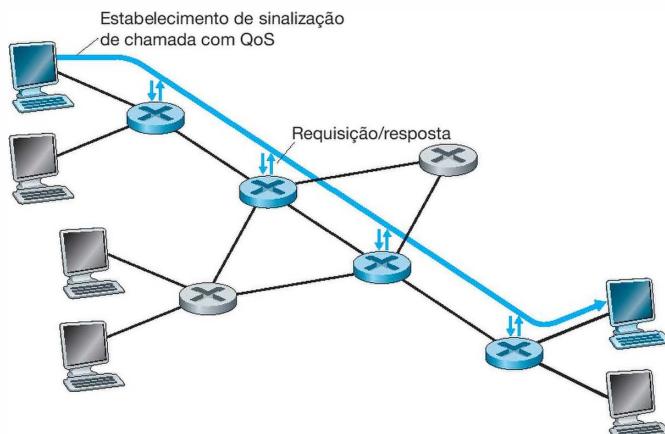
Vamos voltar ao cenário da Seção 7.5.2 e considerar duas aplicações de áudio de 1 Mbit/s que transmitem seus pacotes através do enlace de 1,5 Mbits/s, conforme ilustrado na Figura 7.27. A taxa de dados combinada dos dois fluxos (2 Mbits/s) excede a capacidade do enlace. Mesmo com classificação e marcação, isolamento de fluxos e o compartilhamento de largura de banda não utilizada (que não existe nenhum), este é sem dúvida um caso perdido. Simplesmente não existe largura de banda suficiente para acomodar as necessidades de ambas as aplicações ao mesmo tempo. Se as duas aplicações compartilharem igualmente a largura de banda, cada uma perderia 25% de seus pacotes transmitidos. Essa QoS é tão inaceitavelmente baixa que ambas as aplicações de áudio são completamente inutilizáveis: não há necessidade sequer de transmitir quaisquer pacotes de áudio, em primeiro lugar.

FIGURA 7.27 DUAS APLICAÇÕES DE ÁUDIO CONCORRENTES SOBRECARREGANDO O ENLACE DE R1 A R2

Sabendo que essas duas aplicações, na Figura 7.27, não podem ser atendidas simultaneamente, o que a rede deve fazer? Permitir que as duas prossigam com uma QoS baixa desperdiça os recursos da rede em fluxos de aplicação que, no fim, não oferecem utilidade alguma ao usuário final. A resposta, felizmente, está evidente — um dos fluxos de aplicação deve ser bloqueado (ou seja, deve ter o acesso negado à rede), enquanto o outro deve prosseguir, usando o 1 Mbit/s inteiro necessário pela aplicação. A rede telefônica é um exemplo de uma rede que realiza tal bloqueio de chamada — se os recursos necessários (um circuito fim a fim no caso de uma rede telefônica) não podem ser alocados para a chamada, ela é então bloqueada (impedida de entrar na rede) e o usuário recebe um sinal de ocupado. Em nosso exemplo, não há ganho em permitir que um fluxo entre na rede se ele não vai receber uma QoS suficiente para ser considerada utilizável. De fato, há um custo em admitir um fluxo que não recebe a QoS necessária, pois os recursos da rede estão sendo usados para suportar um fluxo que não oferece utilidade ao usuário final.

Admitindo ou bloqueando de modo explícito os fluxos com base em suas necessidades de recursos e nas necessidades dos fluxos já admitidos, a rede pode garantir que os fluxos admitidos serão capazes de receber sua QoS solicitada. Implícita com a necessidade de prover uma QoS garantida a um fluxo está a necessidade do fluxo de declarar seus requisitos de QoS. O processo de um fluxo declarar seu requisito de QoS e a rede aceitar o fluxo (na QoS solicitada) ou bloqueá-lo é denominado processo de **admissão de chamada**. Este, então, é o nosso quarto princípio (além dos três anteriores da Seção 7.5.2) em relação aos mecanismos necessários para prover a QoS.

Princípio 4: Se recursos suficientes nem sempre estiverem disponíveis e a QoS tiver de ser *garantida,* é necessário um processo de admissão de chamada no qual os fluxos declaram seus requisitos de QoS e, então, são admitidos à rede (na QoS solicitada) ou bloqueados da rede (se a QoS solicitada não puder ser fornecida pela rede).

Nosso exemplo motivador na Figura 7.27 enfatiza a necessidade de diversos novos mecanismos e protocolos de rede, se uma chamada (um fluxo fim a fim) tiver de garantir determinada qualidade de serviço uma vez iniciada:

- *Reserva de recurso.* A única maneira de *garantir* que uma chamada tenha os recursos (largura de banda de enlace, buffers) necessários para satisfazer a QoS desejada é alocar explicitamente esses recursos à chamada — um processo conhecido na linguagem de redes como **reserva de recursos**. Uma vez que os recursos são reservados, a chamada possui acesso sob demanda a esses recursos por toda a sua duração, independentemente das demandas de outras chamadas. Se uma chamada reserva e recebe uma garantia de x Mbits/s de largura de banda de enlace, e nunca transmite a uma taxa maior do que x, a chamada terá um desempenho sem perda e sem atraso.
- *Admissão de chamada.* Se os recursos forem reservados, então a rede deve ter um mecanismo de chamadas para solicitar e reservar recursos. Visto que os recursos não são infinitos, uma chamada que faz uma solicitação de admissão de chamada terá sua admissão negada, ou seja, bloqueada, se os recursos solicitados não estiverem disponíveis. Tal admissão de chamada é realizada pela rede telefônica — so-

licitamos recursos quando discamos um número. Se os circuitos (compartimentos TDMA) necessários para completar a chamada estiverem disponíveis, os circuitos são alocados e a chamada é concluída. Se os circuitos não estiverem disponíveis, então a chamada é bloqueada e recebemos um sinal de ocupado. Uma chamada bloqueada pode tentar novamente ganhar admissão à rede, mas não tem permissão para enviar tráfego à rede até ter completado com sucesso o processo de admissão de chamada. Claro, um roteador que aloca uma largura de banda de enlace não deve alocar mais do que está disponível no enlace. Em geral, uma chamada pode reservar somente uma fração da largura de banda do enlace, e um roteador pode alocar sua largura de banda do enlace para mais de uma chamada. Entretanto, a soma de largura de banda alocada a todas as chamadas deve ser menor do que a capacidade do enlace, se forem dadas garantias fixas de qualidade de serviço.

- *Sinalização do estabelecimento de chamada.* O processo de admissão de chamada descrito acima exige que uma chamada seja capaz de reservar recursos suficientes em todo e qualquer roteador da rede em seu caminho entre os remetentes e os destinatários para garantir que sua exigência da QoS fim a fim seja satisfeita. Cada roteador deve determinar os recursos locais necessários pela sessão, considerar as quantidades de seus recursos que já estão comprometidos com outras sessões em andamento e determinar se ele possui recursos suficientes para satisfazer a exigência da QoS por salto da sessão naquele roteador sem violar as garantias de QoS feitas a uma sessão que já foi admitida. Um protocolo de sinalização é necessário para coordenar essas diversas atividades — a alocação por salto dos recursos locais, bem como a decisão geral de fim a fim de se a ligação foi capaz ou não de reservar recursos necessários em todo e qualquer roteador no caminho fim a fim. Esse é o trabalho do **protocolo de estabelecimento de chamada**, como mostra a Figura 7.28. O **protocolo RSVP** [Zhang, 1993; RFC 2210] foi proposto para essa finalidade dentro da arquitetura da Internet, para oferecer garantias de qualidade de serviço. Em redes ATM, o protocolo Q2931b [Black, 1995] transporta essa informação entre os comutadores da rede ATM e o ponto final.

Apesar de uma tremenda quantidade de pesquisa e desenvolvimento, e até mesmo produtos que oferecem garantias de qualidade de serviço por conexão, quase não tem havido implementação estendida desses serviços. Há muitas razões possíveis. Primeiro, e mais importante, pode ser que os mecanismos simples em nível de aplicação, que estudamos nas seções de 7.2 a 7.4, combinados com o dimensionamento apropriado da rede (Seção 7.5.1), ofereçam um serviço de rede pelo melhor esforço "bom o bastante" para aplicações de multimídia. Além disso, a complexidade e o custo adicional de implementação e gerenciamento de uma rede que ofereça garantias de qualidade de serviço por conexão podem ser considerados pelos ISPs simplesmente como muito altos em comparação com as receitas previstas vindas do cliente para esse serviço.

FIGURA 7.28 O PROCESSO DE ESTABELECIMENTO DE CHAMADA

7.6 RESUMO

A rede multimídia é um dos desenvolvimentos mais interessantes da Internet de hoje. Pessoas em todo o mundo estão passando menos tempo diante de seus aparelhos de rádio ou televisão e usando mais a Internet para receber transmissões de áudio e vídeo, ao vivo ou pré-gravadas. À medida que o acesso à Internet sem fio de alta velocidade se tornar cada vez mais prevalente, essa tendência decerto continuará. Além do mais, com sites do tipo YouTube, os usuários se tornaram produtores e também consumidores de conteúdo de multimídia na Internet. Além da distribuição de vídeo, a Internet está sendo usada para transmitir chamadas telefônicas. De fato, nos próximos dez anos, a Internet, junto com o acesso à Internet sem fio, poderá vir a substituir o quase obsoleto sistema de telefonia de comutação de circuitos. VoIP não somente proverá serviço telefônico mais barato, mas também numerosos serviços de valor agregado, como videoconferência, lista telefônica on-line, serviço de mensagens de voz e integração com redes sociais, como Facebook e Google+.

Na Seção 7.1, descrevemos as características intrínsecas do vídeo e da voz, e depois classificamos as aplicações multimídia em três categorias: (i) áudio/vídeo de fluxo contínuo armazenado, (ii) voz e vídeo-sobre-IP interativos e (iii) áudio e vídeo de fluxo contínuo ao vivo.

Na Seção 7.2, estudamos o vídeo de fluxo contínuo armazenado com mais profundidade. Para aplicações de vídeo de fluxo contínuo, vídeos pré-gravados são armazenados em servidores, e os usuários enviam solicitações a esses servidores para que vejam os vídeos por demanda. Vimos que os sistemas de vídeo de fluxo contínuo podem ser classificados em três categorias: UDP de fluxo contínuo, HTTP de fluxo contínuo e HTT de fluxo contínuo adaptativo. Embora todos esses três tipos de sistemas sejam usados na prática, a maioria dos sistemas de hoje emprega o HTTP de fluxo contínuo e o HTTP de fluxo contínuo adaptativo. Observamos que a medida de desempenho mais importante para o vídeo de fluxo contínuo é a vazão média. Na Seção 7.2, também investigamos as CDNs, que ajudam a distribuir enormes quantidades de dados de vídeo aos usuários do mundo inteiro. Também analisamos a tecnologia por trás de três das principais empresas de vídeo de fluxo contínuo na Internet: Netflix, YouTube e Kankan.

Na Seção 7.3, examinamos como as aplicações de multimídia interativas, como VoIP, podem ser projetadas para trabalhar sobre uma rede de melhor esforço. Para a multimídia interativa, considerações de tempo são importantes, pois as aplicações interativas são altamente sensíveis ao atraso. Por outro lado, aplicações de multimídia interativas são tolerantes a perdas — a perda ocasional só causa lacunas ocasionais na reprodução de áudio/vídeo, e essas perdas em geral podem ser ocultadas parcial ou totalmente. Vimos como uma combinação de buffers do cliente, números de sequência de pacote e marcas de tempo podem aliviar bastante os efeitos da variação de atraso induzido pela rede. Também estudamos a tecnologia por trás da Skype, uma das principais empresas de voz e vídeo sobre IP. Na Seção 7.4, examinamos dois dos protocolos padronizados mais importantes para VoIP, a saber, RTP e SIP.

Na Seção 7.5, introduzimos como vários mecanismos de rede (disciplinas de escalonamento de nível de enlace e regulação de tráfego) podem ser usados para fornecer serviços diferenciados entre diversas classes de tráfego.

EXERCÍCIOS DE FIXAÇÃO E PERGUNTAS

Questões de revisão do Capítulo 7

SEÇÃO 7.1

R1. Reconstrua a Tabela 7.1 para quando Vítor está assistindo um vídeo de 4 Mbits/s, Frank está vendo uma imagem nova de 100 Kbytes a cada 20 segundos e Marta está escutando um fluxo de áudio a 200 kbits/s.

R2. Existem dois tipos de redundância no vídeo. Descreva-os e discuta como eles podem ser explorados para compressão eficiente.

R3. Suponha que um sinal de áudio seja amostrado 16 mil vezes por segundo, e cada amostra seja quantizada em um de 1.024 níveis. Qual seria a taxa de bits resultante do sinal de áudio digital PCM?

R4. Aplicações de multimídia podem ser classificadas em três categorias. Relacione e descreva cada uma dessas categorias.

SEÇÃO 7.2

R5. Sistemas de vídeo de fluxo contínuo podem ser classificados em três categorias. Relacione e descreva de modo resumido cada uma dessas categorias.

R6. Relacione três desvantagens do UDP de fluxo contínuo.

R7. Com o HTTP de fluxo contínuo, o buffer de recepção do TCP e o buffer da aplicação cliente são a mesma coisa? Se não forem, como eles interagem?

R8. Considere o modelo simples para o HTTP de fluxo contínuo. Suponha que o servidor envie bits a uma taxa constante de 2 Mbits/s e a reprodução comece quando 8 milhões de bits tiverem sido recebidos. Qual é o atraso de buffer inicial t_p?

R9. CDNs geralmente adotam uma de duas filosofias de posicionamento de servidor diferentes. Relacione e descreva resumidamente essas duas filosofias.

R10. Diversas estratégias de seleção de *cluster* foram descritas na Seção 7.2.4. Qual dessas estratégias acha um *cluster* bom com relação ao LDNS do cliente? Qual dessas estratégias acha um *cluster* bom com relação ao próprio cliente?

R11. Além das considerações relacionadas à rede, como atraso, perda e desempenho da largura de banda, existem muitos fatores adicionais que entram no projeto de uma estratégia de seleção de *cluster*. Quais são eles?

SEÇÃO 7.3

R12. Qual é a diferença entre atraso fim a fim e variação de atraso do pacote? Quais são as causas da variação de atraso? Quais são as causas da variação de atraso do pacote?

R13. Por que um pacote recebido após seu tempo de reprodução programado é considerado perdido?

R14. Na Seção 7.3, descrevemos dois esquemas FEC. Faça um pequeno resumo deles. Ambos os esquemas aumentam a taxa de transmissão do fluxo adicionando sobrecarga. A intercalação também aumenta a taxa de transmissão?

SEÇÃO 7.4

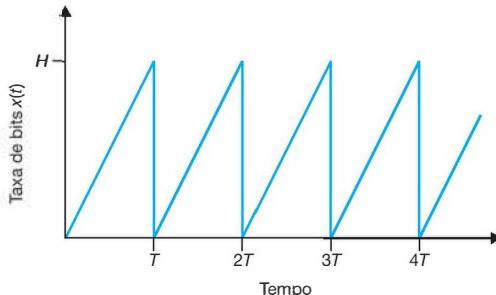
R15. Como os diferentes fluxos RTP em sessões diferentes são identificados por um receptor? Como os diferentes fluxos internos à mesma sessão são identificados?

R16. Qual é o papel de um registro SIP? Qual é a diferença entre o papel de um registro SIP e um agente nativo no IP móvel?

SEÇÃO 7.5

R17. Na Seção 7.5, discutimos o enfileiramento prioritário não preemptivo. O que seria um enfileiramento prioritário preemptivo? O enfileiramento prioritário preemptivo teria sentido em redes de computadores?

R18. Dê um exemplo de disciplina de escalonamento que *não* é conservadora de trabalho.

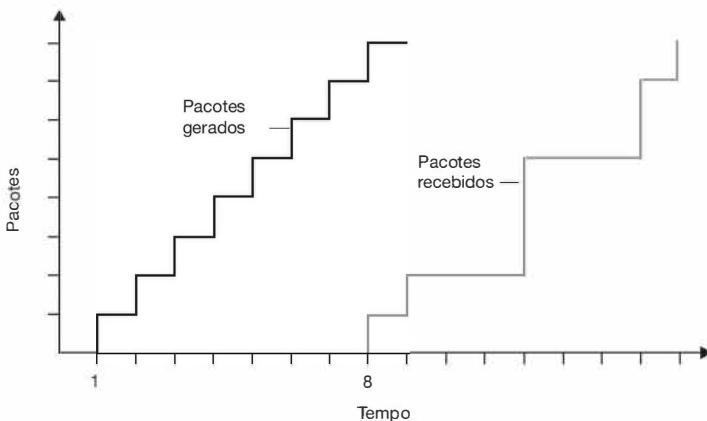
R19. Dê um exemplo de enfileiramentos que você vivencia em sua rotina diária, com disciplina FIFO, prioridade, RR e WFQ.

PROBLEMAS

P1. Considere a figura a seguir. De modo semelhante à nossa discussão da Figura 7.1, suponha que o vídeo seja codificado a uma taxa de bits fixa, e assim cada bloco de vídeo contenha quadros de vídeo que devem ser reproduzidos por algum período de tempo fixo, Δ. O servidor transmite o primeiro bloco de vídeo em t_0, o segundo bloco em $t_0 + \Delta$, o terceiro bloco em t_0 2Δ, e assim por diante. Quando o cliente inicia a reprodução, cada bloco deve ser reproduzido Δ unidades de tempo após o bloco anterior.

a. Suponha que o cliente inicie a reprodução assim que o primeiro bloco chega em t_1. na figura a seguir, quantos blocos de vídeo (incluindo o primeiro) terão chegado ao cliente em tempo para sua reprodução? Explique como você chegou a essa resposta.

b. Suponha que o cliente inicie a reprodução agora em $t_1 + \Delta$. Quantos blocos de vídeo (incluindo o primeiro bloco) terão chegado ao cliente em tempo para sua reprodução? Explique como você chegou a essa resposta.

c. No mesmo cenário do item "b", qual é o maior número de blocos que chega a ser armazenado no buffer do cliente, aguardando a reprodução? Explique como você chegou a essa resposta.

d. Qual é o menor atraso de reprodução no cliente, tal que cada bloco de vídeo tenha chegado em tempo para sua reprodução? Explique como você chegou a essa resposta.

P2. Lembre-se do modelo simples para o HTTP de fluxo contínuo mostrado na Figura 7.3. Lembre-se de que B indica o tamanho do buffer de aplicação do cliente, e Q indica o número de bits que devem ser mantidos em buffer antes que a aplicação cliente inicie a reprodução. Além disso, r indica a taxa de consumo de vídeo. Suponha que o servidor envie bits a uma taxa constante x sempre que o buffer do cliente não está cheio.

a. Suponha que $x < r$. Conforme discutimos no texto, neste caso a reprodução alternará entre períodos de reprodução contínua e períodos de congelamento. Determine a extensão de cada reprodução contínua e período de congelamento como uma função de Q, r e x.

b. Agora, suponha que $x > r$. Em que momento $t = t_f$ o buffer da aplicação cliente se torna cheio?

P3. Lembre-se do modelo simples para o HTTP de fluxo contínuo mostrado na Figura 7.3. Suponha que o tamanho do buffer seja infinito, mas o servidor envie bits na taxa variável $x(t)$. Especificamente, suponha que $x(t)$ tenha a seguinte forma de dente de serra. A taxa é de início zero no instante $t = 0$ e sobe de modo linear até H no instante $t = T$. Depois, esse padrão é repetido indefinidamente, como mostra a figura a seguir.

a. Qual é a taxa de envio média do servidor?

b. Suponha que $Q = 0$, de modo que o cliente começa a reproduzir assim que recebe um quadro de vídeo. O que acontecerá?

c. Agora, suponha que $Q > 0$. Determine, como uma função de Q, H e T, o instante em que a reprodução inicia.

d. Suponha que $H > 2r$ e $Q = HT/2$. Prove que não haverá congelamento após o atraso de reprodução inicial.

e. Suponha que $H > 2r$. Ache o menor valor de Q tal que não haverá congelamento após o atraso de reprodução inicial.

f. Agora, suponha que o tamanho do buffer B seja finito. Considere que $H > 2r$. Como uma função de Q, B, T e H, determine o instante $t = t_f$ em que o buffer da aplicação cliente se torna cheio inicialmente.

P4. Lembre-se do modelo simples para o HTTP de fluxo contínuo mostrado na Figura 7.3. Suponha que o buffer da aplicação cliente seja infinito, o servidor envie à taxa constante x e a taxa de consumo de vídeo seja r com $r < x$. Considere também que a reprodução comece imediatamente. Suponha que o usuário termine o vídeo antecipadamente, no instante $t = E$. No momento do término, o servidor para de enviar bits (se ainda não tiver enviado todos os bits do vídeo).

a. Suponha que o vídeo seja infinitamente longo. Quantos bits são desperdiçados (isto é, enviados, mas não vistos)?

b. Suponha que o vídeo tenha T segundos de duração com $T > E$. Quantos bits são desperdiçados (isto é, enviados, mas não vistos)?

P5. Considere um sistema DASH para o qual existem N versões de vídeo (em N diferentes taxas e qualidades) e N versões de áudio (em N taxas e versões diferentes). Suponha que queiramos permitir que o dispositivo de reprodução escolha, a qualquer momento, qualquer uma das N versões de vídeo e qualquer uma das N versões de áudio.

a. Se criarmos arquivos de modo que o áudio seja misturado com o vídeo, de modo que o servidor envia somente um fluxo de mídia em determinado momento, quantos arquivos o servidor precisará armazenar (cada um com um URL diferente)?

b. Se o servidor, em vez disso, envia os fluxos de áudio e vídeo separadamente e o cliente sincroniza os fluxos, quantos arquivos o servidor precisa armazenar?

P6. No exemplo de VoIP da Seção 7.3, seja h o número total de bytes de cabeçalho adicionados a cada trecho, incluindo o cabeçalho UDP e IP.

a. Supondo que um datagrama IP seja emitido a cada 20 ms, ache a taxa de transmissão em bits por segundo para os datagramas gerados por um lado dessa aplicação.

b. Qual é um valor típico de h quando o RTP é usado?

P7. Considere o procedimento descrito na Seção 7.3 para estimar o atraso médio d_i. Suponha que $u = 0{,}1$. Seja $r_1 - t_1$ o atraso da amostra mais recente, seja $r_2 - t_2$ o atraso da amostra mais recente seguinte e assim por diante.

a. Para uma dada aplicação de áudio, suponha que quatro pacotes tenham chegado ao receptor com atrasos de amostra de $r_4 - t_4$, $r_3 - t_3$, $r_2 - t_2$ e $r_1 - t_1$. Expresse a estimativa de atraso d em termos das quatro amostras.

b. Generalize sua fórmula para n atrasos de amostra.

c. Para a fórmula do item "b", suponha que n tenda ao infinito e dê a fórmula resultante. Comente por que esse procedimento de determinação de média é denominado média móvel exponencial.

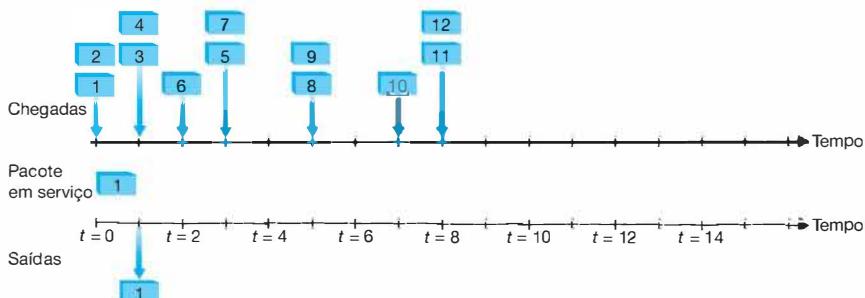
P8. Repita os itens "a" e "b" da questão anterior para a estimativa do desvio médio do atraso.

P9. No exemplo de VoIP na Seção 7.3 apresentamos um procedimento on-line (média móvel exponencial) para estimar o atraso. Neste problema examinaremos um procedimento alternativo. Seja t_i a marca de tempo do *i*-ésimo pacote recebido; r_i o tempo em que o *i*-ésimo pacote é recebido. Seja d_n nossa estimativa do atraso médio após o recebimento do *enésimo* pacote. Após a recepção do primeiro pacote, estabelecemos a estimativa de atraso como $d_1 = r_1 - t_1$.

a. Suponha que gostaríamos que $d_n = (r_1 - t_1 + r_2 - t_2 + ... + r_n - t_n)/n$ para todo *n*. Deduza uma fórmula recursiva para d_n em termos de d_{n-1}, r_n e t_n.

b. Descreva por que, para a telefonia por Internet, a estimativa de atraso descrita na Seção 7.3 é mais apropriada do que a esboçada no item "a".

P10. Compare o procedimento descrito na Seção 7.3 para a estimativa do desvio médio do atraso com o procedimento descrito na Seção 3.5 para a estimativa do tempo de ida e volta. O que os procedimentos têm em comum? Em que são diferentes?

P11. Considere a figura a seguir (que é semelhante à Figura 7.7). Um transmissor começa a enviar áudio empacotado periodicamente em $t = 1$. O primeiro pacote chega quando o transmissor está em $t = 8$.

a. Quais são os atrasos (do transmissor ao receptor, ignorando qualquer atraso de transmissão) dos pacotes de 2 a 8? Observe que cada segmento de linha vertical e horizontal na figura tem o comprimento de 1, 2 ou 3 unidades de tempo.

b. Se a reprodução de áudio começa assim que o primeiro pacote chega no receptor em $t = 8$, qual dos primeiros oito pacotes enviados *não* chegará em tempo para a reprodução?

c. Se a reprodução de áudio começa em $t = 9$, qual dos primeiros oito pacotes enviados não chegará em tempo para a reprodução?

d. Qual o mínimo de atraso de reprodução no receptor que resulta em todos os primeiros oito pacotes chegando a tempo para a reprodução?

P12. Considere novamente a figura em P11, que mostra os tempos de transmissão dos pacotes de áudio e os tempos de recepção.

a. Calcule o atraso estimado para os pacotes de 2 a 8, usando a fórmula para d_i, da Seção 7.3.2. Use o valor de $u = 0{,}1$.

b. Calcule o desvio estimado do atraso para a média estimada para os pacotes 2 a 8, usando a fórmula para v_i da Seção 7.3.2. Use o valor $u = 0{,}1$.

P13. Lembre-se dos dois esquemas FEC para VoIP descritos na Seção 7.3. Suponha que o primeiro esquema gere um trecho redundante para cada quatro trechos originais. Suponha que o segundo use uma codificação de baixa taxa de bits, cuja taxa de transmissão seja 25% da taxa de transmissão do fluxo nominal.

a. Quanta largura de banda adicional cada esquema requer? E quanto atraso de reprodução cada esquema adiciona?

b. Como os dois esquemas funcionarão se, em cada grupo de cinco pacotes, o primeiro for perdido? Qual esquema terá melhor qualidade de áudio?

c. Como os dois esquemas funcionarão se, em cada grupo de dois pacotes, o primeiro for perdido? Qual esquema terá melhor qualidade de áudio?

P14. a. Considere uma chamada de audioconferência no Skype com $N > 2$ participantes. Suponha que cada participante gera um fluxo constante com taxa de r bits/s. Quantos bits por segundo o iniciador da chamada precisa enviar? Quantos bits por segundo cada um dos outros $N - 1$ participantes precisará enviar? Qual é a taxa de envio total, agregada por todos os participantes?

b. Repita o item "a" para uma chamada de videoconferência no Skype usando um servidor central.

c. Repita o item "b", mas agora para quando cada par enviar uma cópia de seu fluxo de vídeo a cada um dos $N - 1$ outros pares.

P15. a. Suponha que enviemos dois datagramas IP para a Internet, cada um portando um segmento UDP diferente. O primeiro tem endereço IP de origem A1, endereço IP de destino B, porta de origem P1 e porta de destino T. O segundo tem endereço IP de origem A2, endereço IP de destino B, porta de origem P2 e porta de destino T. Suponha que A1 é diferente de A2 e P1 é diferente de P2. Admitindo que ambos os datagramas cheguem a seu destino final, os dois datagramas UDP serão recebidos pelo mesmo *socket?* Justifique sua resposta.

b. Suponha que Alice, Bob e Claire queiram fazer uma audioconferência usando SIP e RTP. Para Alice enviar e receber pacotes RTP de e para Bob e Claire, somente uma porta UDP é suficiente (além da necessária para as mensagens SIP)? Caso a resposta seja positiva, então como o cliente SIP de Alice distingue entre os pacotes RTP recebidos de Bob e Claire?

P16. Verdadeiro ou falso:

a. Se um vídeo armazenado é entregue diretamente de um servidor Web a um reprodutor de mídia, então a aplicação está usando TCP como protocolo de transporte subjacente.

b. Ao usar RTP, é possível que um remetente mude a codificação no meio de uma sessão.

c. Todas as aplicações que usam RTP devem usar a porta 87.

d. Suponha que uma sessão RTP tenha fluxos separados de áudio e vídeo para cada remetente. Então, os fluxos de áudio e vídeo usam o mesmo SSRC.

e. Em serviços diferenciados, ainda que o comportamento por salto defina diferenças de desempenho entre classes, ele não impõe nenhum mecanismo particular para alcançar esses desempenhos.

f. Suponha que Alice queira estabelecer uma sessão SIP com Bob. Ela inclui, na sua mensagem INVITE, a linha m=audio 48753 RTP/AVP 3 (AVP 3 indica áudio GSM). Portanto, Alice indicou em sua mensagem que ela deseja enviar áudio GSM.

g. Com referência à declaração anterior, Alice indicou em sua mensagem INVITE que enviará áudio para a porta 48753.

h. Mensagens SIP são enviadas tipicamente entre entidades SIP usando um número *default* para a porta SIP.

i. Para manter seu registro, clientes SIP têm de enviar mensagens REGISTER periodicamente.

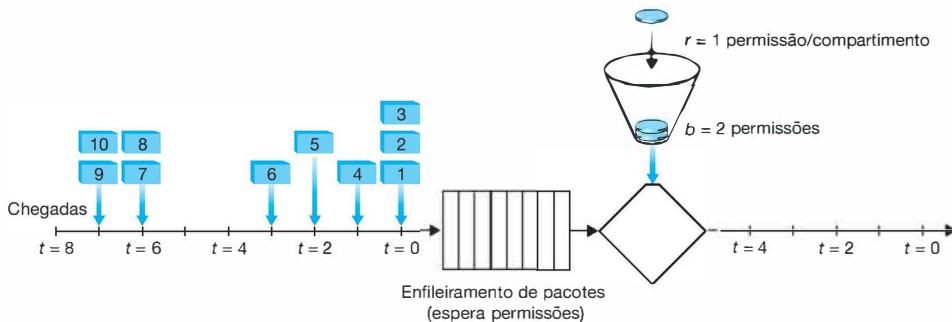
j. SIP impõe que todos os clientes SIP suportem codificação de áudio G.711.

P17. Suponha que a política de escalonamento WFQ seja aplicada a um buffer que suporta três classes; suponha que os pesos para essas três classes sejam 0,5, 0,25 e 0,25.

a. Suponha que cada classe tenha um grande número de pacotes no buffer. Em que sequência poderiam

ser atendidas essas três classes para atingir os pesos WFQ descritos? (Para escalonamento por varredura cíclica, uma sequência natural é 123123123...).

b. Suponha que as classes 1 e 2 tenham um grande número de pacotes no buffer e que não haja pacotes de classe 3 no buffer. Em que sequência as três classes poderiam ser atendidas para alcançar os pesos WFQ descritos?

P18. Considere a figura a seguir. Responda as seguintes perguntas:

a. Supondo que o serviço é FIFO, indique o tempo em que os pacotes de 2 a 12 deixam a fila. Para cada pacote, qual o atraso entre a chegada e o início do compartimento no qual é transmitido? Qual o atraso médio sobre todos os 12 pacotes?

b. Suponha agora um serviço com prioridades, e admita que os números ímpares são de prioridade alta, e os pares de prioridade baixa. Indique o tempo em que cada pacote de 2 a 12 deixará a fila. Para cada pacote, qual o atraso entre a chegada e o início do compartimento no qual é transmitido? Qual o atraso médio sobre todos os 12 pacotes?

c. Suponha agora um serviço de varredura cíclica, e admita que os pacotes 1, 2, 3, 6, 11 e 12 sejam de classe 1, e os pacotes 4, 5, 7, 8, 9 e 10 de classe 2. Indique o tempo em que cada pacote de 2 a 12 deixará a fila. Para cada um, qual o atraso entre a chegada e a partida? Qual o atraso médio para todos os 12 pacotes?

d. Suponha agora a disciplina de serviço de enfileiramento justo ponderado (WFQ), e admita que os números ímpares são de prioridade alta, e os pares de prioridade baixa, a partir da classe 2. A classe 1 tem um peso WFQ de 2, enquanto a classe 2 tem um peso WFQ de 1. Observe que pode não ser possível alcançar uma sincronização WFQ idealizada como vimos no texto, então indique por que você escolheu o pacote específico para ser atendido em cada compartimento de tempo. Para cada pacote, qual é o atraso entre a chegada e a partida? Qual é o atraso médio para todos os 12 pacotes?

e. O que você pode observar sobre o tempo de atraso médio nos quatro casos (FIFO, RR, prioridade e WFQ)?

P19. Considere novamente a figura de P18.

a. Suponha um serviço prioritário, com os pacotes 1, 4, 5, 6 e 11 sendo de prioridade alta. Os pacotes restantes são de prioridade baixa. Indique os compartimentos nos quais cada pacote de 2 a 12 deixará a fila.

b. Agora suponha que um serviço de varredura cíclica seja usado, com os pacotes 1, 4, 5, 6 e 11 pertencentes a uma classe de tráfego, e os restantes pertencendo a uma segunda classe de tráfego. Indique os compartimentos nos quais cada pacote de 2 a 11 deixará a fila.

c. Suponha agora um serviço WFQ, com os pacotes 1, 4, 5, 6 e 11 pertencentes a uma classe de tráfego, e os restantes pertencendo a uma segunda classe de tráfego. A classe 1 tem um peso WFQ de 1, enquanto a classe 2 tem um peso WFQ de 2 (observe que estes pesos são diferentes daqueles na questão anterior). Indique os compartimentos nos quais cada pacote de 2 a 12 deixará a fila. Veja também a advertência na questão anterior relativa ao serviço WFQ.

P20. Considere a figura abaixo, que mostra um regulador de balde furado sendo alimentado por um fluxo contínuo de pacotes. O buffer de permissões pode segurar, no máximo, duas permissões e está inicialmente cheio em $t = 0$. Novas permissões chegam em uma velocidade de uma permissão por compartimento. A velocidade do enlace de saída é tal que se dois pacotes obtêm permissões no início de um compartimento de tempo, ambos podem ir ao enlace de saída no mesmo compartimento. Os detalhes de temporização são os seguintes:

1. Os pacotes (se houver) chegam no começo do compartimento. Sendo assim, na figura, os pacotes 1, 2 e 3 chegam no compartimento 0. Se já existirem pacotes na fila, os que vão chegar se juntam no final da fila. Os pacotes prosseguem em direção ao começo da fila de modo FIFO.
2. Depois das chegadas serem adicionadas à fila, se já existirem pacotes, um ou dois deles (dependendo do número de permissões disponíveis) removerão uma permissão do buffer de permissões e irão ao enlace de saída durante esse compartimento. Sendo assim, os pacotes 1 e 2 removem uma permissão do buffer (já que existem inicialmente duas permissões) e vão ao enlace de saída durante o compartimento 0.
3. Uma nova permissão é adicionada ao buffer de permissões se este não estiver cheio, já que a velocidade de criação de permissões é de $r = 1$ permissão/compartimento.
4. Tempo avança até a próxima entrada, e as etapas se repetem.

Responda às seguintes perguntas:

a. Para cada compartimento de tempo, identifique os pacotes que estão na fila e o número de permissões no balde, imediatamente depois que as chegadas foram processadas (etapa 1 acima) mas antes que os pacotes passem pela fila e removam uma permissão. Assim, para o compartimento de tempo $t = 0$ no exemplo acima, os pacotes 1, 2 e 3 estão na fila, e existem duas permissões no buffer.
b. Para cada compartimento de tempo, indique qual pacote aparecerá na saída depois que as permissões forem removidas da fila. Assim, para o compartimento $t = 0$ no exemplo acima, os pacotes 1 e 2 aparecem no enlace de saída do balde furado durante a entrada 0.

P21. Repita o problema P20, mas suponha que $r = 2$. Admita novamente que o balde está cheio inicialmente.

P22. Considere o problema P21, mas suponha agora que $r = 3$, e que $b = 2$, como antes. Sua resposta para a questão acima é diferente?

P23. Considere o regulador de balde furado, que regula a taxa média e o tamanho da rajada de um fluxo de pacotes. Queremos agora também regular a taxa de pico p. Mostre como a saída desse regulador de balde furado pode ser alimentada para um segundo regulador de balde furado, de modo que os dois, em série, regulem a taxa média, a taxa de pico e o tamanho da rajada. Não se esqueça de atribuir um tamanho e uma taxa de geração de permissão ao balde do segundo regulador.

P24. Diz-se que um fluxo de pacotes está de acordo com a especificação de um balde furado (r,b) com tamanho de rajada b e taxa média r se o número de pacotes que chega ao balde furado for menor do que $rt + b$ pacotes a cada intervalo de tempo de duração t para todo t. Um fluxo de pacotes que esteja de acordo com a especificação do balde furado (r,b) alguma vez terá de esperar em um regulador de balde furado com parâmetros r e b? Justifique sua resposta.

P25. Demonstre que, enquanto $r_1 < R\, w_1/(\sum w_j)$, então d_{max} é, na verdade, o atraso máximo que qualquer pacote do fluxo 1 sofrerá na fila WFQ.

TAREFA DE PROGRAMAÇÃO

Neste laboratório você implementará um servidor e um cliente de vídeo de fluxo contínuo. O cliente usará o protocolo de fluxo contínuo em tempo real (RTSP) para controlar as ações do servidor. O servidor usará o protocolo de tempo real (RTP) para empacotar o vídeo para transporte por UDP. Você receberá um código Python que executará parcialmente RTSP e RTP no cliente e no servidor. Sua tarefa será concluir o código para o cliente e também para o servidor. Quando terminar, você terá criado uma aplicação cliente-servidor que faz o seguinte:

- O cliente envia comandos RTSP SETUP, PLAY, PAUSE e TEARDOWN e o servidor responde aos comandos.
- Quando o servidor estiver no estado de reprodução, ele pega periodicamente um quadro JPEG armazenado, empacota o quadro com RTP e envia o pacote RTP para um *socket* UDP.
- O cliente recebe os pacotes RTP, extrai os quadros JPEG, descompacta os quadros e os apresenta no monitor do cliente.

O código que você receberá implementa o protocolo RTSP no servidor e o desempacotamento RTP no cliente, e também cuida da apresentação do vídeo transmitido. Você precisará executar RTSP no cliente e RTP no servidor. Esta tarefa de programação aprimorará de modo significativo a compreensão de RTP, RTSP e vídeo de fluxo contínuo para o aluno. Recomendamos veementemente que ela seja executada. A tarefa também sugere vários exercícios opcionais, incluindo implementação do comando RTSP DESCRIBE no cliente e também no servidor. Você encontrará informações completas sobre a tarefa, bem como trechos importantes de código Python, no site de apoio deste livro.

ENTREVISTA

Henning Schulzrinne

Henning Schulzrinne é professor, diretor do Departamento de Ciências da Computação e chefe do Internet Real-Time Laboratory da Universidade Columbia. É coautor dos protocolos RTP, RTSP, SIP e GIST, fundamentais para comunicação de áudio e vídeo pela Internet. É bacharel em engenharia elétrica e industrial pela Universidade de Darmstadt, na Alemanha, mestre em engenharia elétrica e de computação pela Universidade de Cincinnati e doutor em engenharia elétrica pela Universidade de Massachusetts, em Amherst.

O que o fez se decidir pela especialização em tecnologia de redes multimídia?

Aconteceu quase por acidente. Quando fazia doutorado, acabei me envolvendo com a DARTnet, uma rede experimental que abrangia os Estados Unidos com linhas T1. A DARTnet era usada como campo de prova para ferramentas de transmissão para um grupo (*multicast*) e de Internet em tempo real. Isso me levou

a desenvolver minha primeira ferramenta de áudio, a NeVoT. Por intermédio de alguns participantes da DARTnet, acabei me envolvendo com a IETF, com o grupo de trabalho Audio Vídeo Transport, que se formara havia pouco tempo naquela época. O grupo terminou por padronizar o RTP.

Qual foi seu primeiro emprego no setor de computação? O que implicava?

No meu primeiro emprego no setor de computação fiz a soldagem das peças de um computador Altair. Eu ainda era estudante do ensino médio em Livermore, na Califórnia. Quando voltei à Alemanha, montei uma pequena empresa de consultoria que projetou um programa de gerenciamento de endereços para uma agência de viagens — armazenando dados em fitas cassete para nosso TRS-80 e usando uma máquina de escrever elétrica Selectric da IBM com uma interface de hardware feita em casa servindo de impressora.

Meu primeiro emprego verdadeiro foi no AT&T Bell Laboratories, no desenvolvimento de um emulador de redes para a construção de redes experimentais em ambiente de laboratório.

Quais os objetivos do Internet Real-Time Lab?

Nosso objetivo é fornecer componentes e construir blocos para a Internet como uma única infraestrutura de comunicação. Isso inclui o desenvolvimento de novos protocolos, como o GIST (para sinalização de camadas de rede) e o LoST (para encontrar recursos por localização), ou aprimorar os protocolos nos quais já trabalhamos anteriormente, como o SIP, por meio de trabalhos de presença aprimorada, sistemas *peer-to-peer*, chamadas de emergência de próxima geração e ferramentas de criação de serviços. Recentemente, também voltamos nossa atenção aos sistemas sem fio VoIP, como as redes 802.11b e 802.11n, e talvez as redes WiMax se tornem importantes tecnologias de última milha para telefonia. Também estamos tentando aprimorar a capacidade de os usuários diagnosticarem falhas nos confusos equipamentos e provedores, usando um sistema de diagnóstico de falhas *peer-to-peer* chamado DYSWIS (Do You See What I See [você vê o que eu vejo]).

Tentamos fazer trabalhos relevantes, ao construir protótipos e sistemas de código-fonte aberto, ao medir o desempenho de sistemas reais, e contribuindo para os padrões da IETF.

Em sua opinião, qual é o futuro das redes multimídia?

Estamos agora em uma fase de transição; poucos anos nos separam do IP como plataforma universal para serviços de multimídia, do IPTV ao VoIP. Esperamos que o rádio, o telefone e a TV continuem funcionando mesmo durante tempestades de neve e terremotos; portanto, quando a Internet assumir os papéis dessas redes dedicadas, os usuários poderão esperar o mesmo nível de confiabilidade.

Teremos de aprender a projetar tecnologias de rede em um ecossistema de operadoras, provedores de serviço e conteúdo concorrentes, servindo muitos usuários sem treinamento e defendendo-os de um pequeno, porém destrutivo, punhado de usuários maliciosos e criminosos. A mudança de protocolos está se tornando cada vez mais difícil. Estão se tornando também mais complexas, já que precisam levar em conta os lucros competitivos de negócios, segurança, privacidade e a falta de transparência das redes, causada por *firewalls* e tradutores de endereços de rede.

Desde que a rede multimídia se tornou a base para todo entretenimento ao consumidor, existirá uma ênfase na administração de grandes redes, a um preço baixo. Os usuários esperarão um uso fácil, como ter o mesmo conteúdo em todos seus dispositivos.

Por que o SIP tem um futuro promissor?

Enquanto prossegue a evolução das redes sem fio existentes hoje para 3G, há a esperança de um único mecanismo de sinalização para multimídia que abranja todos os tipos de rede, desde modem a cabo até redes telefônicas de empresas e redes públicas sem fio. Junto com softwares de rádio, isso possibilitará que, no futuro, um dispositivo único, tal como um telefone Bluetooth sem fio, possa ser utilizado em uma rede residencial, em uma rede empresarial via 802.11 e em redes de longa distância por meio de redes 3G. Mesmo antes de existir tal dispositivo único, universal, sem fio, os mecanismos de mobilidade pessoal permitem ocultar as diferenças entre redes. Um identificador torna-se o meio universal de alcançar uma pessoa, em vez de ter de lembrar ou ter de passar por meia dúzia de números de telefones específicos para tecnologias ou localizações.

O SIP também separa o fornecimento de transporte de voz (bit) dos serviços de voz. Agora ficou tecnicamente possível acabar com o monopólio local de telefonia, no qual uma única empresa fornece transporte neutro de

bits enquanto outras fornecem "o tom de discar" IP e os serviços clássicos de telefonia, como gateways, transferência de chamadas e identificador de chamadas.

Além da sinalização de multimídia, o SIP oferece um novo serviço que estava faltando na Internet: notificação de eventos. Algo parecido com esse serviço já vem sendo oferecido por soluções improvisadas HTTP e por e-mail, mas nunca foi muito satisfatório. Uma vez que eventos são uma abstração comum para sistemas distribuídos, isso pode simplificar a construção de novos serviços.

Você pode dar algum conselho aos estudantes que estão ingressando no campo das redes?

O trabalho com redes é quase interdisciplinar. Ele tira subsídios da engenharia elétrica, da ciência da computação, da pesquisa operacional e de outras disciplinas. Assim, os pesquisadores de redes têm de estar familiarizados com assuntos que estão além de protocolos e algoritmos de roteamento.

Já que as redes estão se tornando partes tão importantes do nosso dia a dia, estudantes que procuram fazer a diferença no campo deveriam pensar em limitações de novos recursos em rede: tempo e esforço humano, em vez de banda larga e armazenamento.

O trabalho com redes pode trazer imensa satisfação, já que se trata de permitir que as pessoas se comuniquem e troquem ideias, um dos valores essenciais do ser humano. A Internet se tornou a terceira maior infraestrutura global, seguindo o sistema de transporte e a distribuição de energia. Poucos setores da economia conseguem trabalhar sem redes de alto desempenho, então deverão existir muitas oportunidades para o futuro próximo.

CAPÍTULO

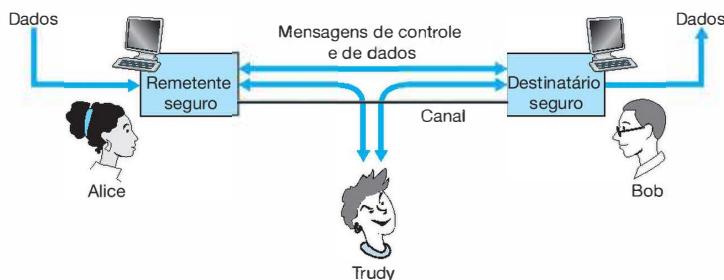
8

SEGURANÇA EM REDES DE COMPUTADORES

Na Seção 1.6 descrevemos algumas das categorias mais predominantes e prejudiciais dos ataques na Internet, incluindo ataques *malware*, recusa de serviço, analisador de pacotes, disfarce da origem e modificação e exclusão de mensagem. Embora tenhamos aprendido muito sobre redes de computadores, ainda não analisamos como protegê-las dos ataques descritos na Seção 1.6. Com nosso conhecimento recém-adquirido em rede de computadores e protocolos da Internet, estudaremos minuciosamente a comunicação segura e, em particular, como as redes podem ser protegidas desses vilões.

Queremos lhe apresentar Alice e Bob, duas pessoas que desejam se comunicar, porém "com segurança". Como esse texto refere-se a redes, gostaríamos de observar que Alice e Bob podem ser dois roteadores que querem trocar tabelas de roteamento com segurança, um cliente e um servidor que querem estabelecer uma conexão de transporte segura ou duas aplicações de e-mail que querem trocar e-mails com segurança — todos esses tópicos serão examinados mais adiante neste capítulo. Alice e Bob são componentes conhecidos da comunidade de segurança, talvez porque o nome deles seja mais interessante do que uma entidade genérica denominada "A" que quer se comunicar com segurança com uma entidade genérica denominada "B". Amores proibidos, comunicações em tempo de guerra e transações financeiras são as necessidades dos seres humanos mais citadas quando o assunto é segurança nas comunicações; preferimos a primeira necessidade às duas últimas, e vamos usar, com muito prazer, Alice e Bob como nosso remetente e nosso destinatário e imaginá-los nesse primeiro cenário.

Dissemos que Alice e Bob querem se comunicar, porém "com segurança", mas o que isso significa exatamente? Como veremos, a segurança (assim como o amor) é repleta de maravilhas; isto é, ela tem muitas facetas. É certo que Alice e Bob gostariam que o conteúdo de sua comunicação permanecesse secreto, a salvo de um bisbilhoteiro. Provavelmente, eles também gostariam de ter certeza de que estão se comunicando mesmo um com o outro e de que, caso algum bisbilhoteiro interfira na comunicação, essa interferência seja detectada. Na primeira parte deste capítulo, estudaremos as técnicas que permitem criptografar/decriptar comunicações, autenticar a parte com quem estamos nos comunicando e assegurar a integridade da mensagem.

Na segunda parte, examinaremos como os princípios da criptografia podem ser usados para criar protocolos de rede seguros. Utilizando mais uma vez uma abordagem *top-down*, examinaremos os protocolos seguros em cada uma das (quatro principais) camadas, iniciando pela camada de aplicação. Verificaremos como proteger o e-mail e uma conexão TCP, como prover segurança total na camada de rede, e como proteger uma LAN sem fio. Na terceira parte deste capítulo, avaliaremos a segurança operacional, que tem o objetivo de proteger redes organizacionais de ataques. Em particular, verificaremos, de forma minuciosa, como os *firewalls* e os sistemas de detecção de invasão podem aprimorar a segurança de uma rede organizacional.

8.1 O QUE É SEGURANÇA DE REDE?

Vamos iniciar nosso estudo de segurança de redes voltando aos namorados citados, Alice e Bob, que querem se comunicar "com segurança". O que isso significa ao certo? Com certeza, Alice quer que somente Bob entenda a mensagem que ela enviou, mesmo que eles estejam se comunicando por um meio inseguro, em que um intruso (Trudy, a intrusa) pode interceptar qualquer dado que seja transmitido. Bob também quer ter certeza de que a mensagem que recebe de Alice foi de fato enviada por ela, e Alice quer ter certeza de que a pessoa com quem está se comunicando é de fato Bob. Alice e Bob também querem ter certeza de que o conteúdo de suas mensagens não foi alterado em trânsito. Também querem, antes de tudo, ter certeza de que podem se comunicar (isto é, de que ninguém lhes negue acesso aos recursos necessários à comunicação). Dadas essas considerações, podemos identificar as seguintes propriedades desejáveis da **comunicação segura**:

- *Confidencialidade.* Apenas o remetente e o destinatário pretendido devem poder entender o conteúdo da mensagem transmitida. O fato de intrusos conseguirem interceptar a mensagem exige, necessariamente, que esta seja **cifrada** de alguma maneira para impedir que seja entendida por um interceptador. Esse aspecto de confidencialidade é, provavelmente, o significado mais comumente percebido na expressão *comunicação segura*. Estudaremos técnicas de criptografia para cifrar e decifrar dados na Seção 8.2.
- *Integridade de mensagem.* Alice e Bob querem assegurar que o conteúdo de sua comunicação não seja alterado, por acidente ou por má intenção, durante a transmissão. Extensões das técnicas de soma de verificação que encontramos em protocolos de transporte e de enlace confiáveis podem ser utilizadas para proporcionar integridade à mensagem. Estudaremos autenticação do ponto de chegada e integridade da mensagem na Seção 8.3.
- *Autenticação do ponto final.* O remetente e o destinatário precisam confirmar a identidade da outra parte envolvida na comunicação — confirmar que a outra parte é de verdade quem alega ser. A comunicação pessoal entre seres humanos resolve facilmente esse problema por reconhecimento visual. Quando entidades comunicantes trocam mensagens por um meio pelo qual não podem ver a outra parte, a autenticação não é assim tão simples. Por que, por exemplo, você deveria acreditar que o e-mail que recebeu e que contém uma sentença afirmando que aquele e-mail veio de um amigo seu vem mesmo dele? Estudamos a autenticação do ponto final na Seção 8.4.
- *Segurança operacional.* Hoje quase todas as organizações (empresas, universidades etc.) possuem redes conectadas à Internet pública. Essas redes podem ser comprometidas por atacantes que ganham acesso a elas por meio da Internet pública. Os atacantes podem tentar colocar *worms* nos hospedeiros na rede, adquirir segredos corporativos, mapear as configurações internas da rede e lançar ataques de DoS. Veremos na Seção 8.9 que os mecanismos operacionais, como *firewalls* e sistemas de detecção de invasão, são usados para deter ataques contra a rede de uma organização. Um *firewall* localiza-se entre a rede da organização e a rede pública, controlando os acessos de pacote de e para a rede. Um sistema de detecção de invasão realiza uma "profunda inspeção de pacote", alertando os administradores da rede sobre alguma atividade suspeita.

Agora que já determinamos o que significa segurança de rede, vamos considerar em seguida quais são, exatamente, as informações às quais um intruso pode ter acesso e que ações podem ser executadas por ele. A Figura 8.1 ilustra o cenário. Alice, a remetente, quer enviar dados a Bob, o destinatário. Para trocar dados com segurança, além de atender aos requisitos de confidencialidade, autenticação e integridade de mensagens, Alice e Bob trocarão mensagens de controle e de dados (algo muito semelhante ao modo como remetentes e destinatários TCP trocam segmentos de controle e segmentos de dados). Todas ou algumas dessas mensagens costumam ser criptografadas. Conforme discutimos na Seção 1.6, um intruso passivo consegue, potencialmente, fazer o seguinte:

- *monitorar* — identificar e gravar as mensagens de controle e de dados no canal;
- *modificar, inserir ou eliminar* mensagens ou conteúdo de mensagens.

FIGURA 8.1 REMETENTE, DESTINATÁRIO E INTRUSO (ALICE, BOB E TRUDY)

Como veremos, a menos que sejam tomadas contramedidas adequadas, essas capacidades permitem que um intruso monte uma grande variedade de ataques à segurança: monitorar comunicações (talvez roubando senhas e dados), fazer-se passar por outra entidade, sequestrar uma sessão em curso, recusar serviço a usuários legítimos da rede sobrecarregando os recursos do sistema e assim por diante. O CERT Coordination Center [CERT, 2012] mantém um resumo de ataques comunicados.

Agora que já temos certeza de que há ameaças reais à solta na Internet, quais são os equivalentes de Alice e Bob na Internet, esses nossos amigos que precisam se comunicar com segurança? Decerto, Bob e Alice podem ser dois usuários humanos em dois sistemas finais, por exemplo, uma Alice real e um Bob real que de fato querem trocar e-mails seguros. Eles podem também ser os participantes de uma transação de comércio eletrônico. Por exemplo, um Bob real pode querer transmitir com segurança o número de seu cartão de crédito a um servidor Web para comprar um produto pela rede. As partes que necessitam de uma comunicação segura podem fazer parte de uma infraestrutura da rede. Lembre-se de que o sistema de nomes de domínio (DNS, veja a Seção 2.5) ou *daemons* roteadores que trocam informações de roteamento (veja a Seção 4.6) requerem comunicação segura entre dois participantes. O mesmo é válido para aplicações de gerenciamento de rede, um tópico que examinaremos no Capítulo 9. Um intruso que conseguisse interferir em consultas do DNS (como discutido na Seção 2.5), processamentos de roteamento [RFC 4272] ou funções de gerenciamento de rede [RFC 3414] poderia causar uma devastação na Internet.

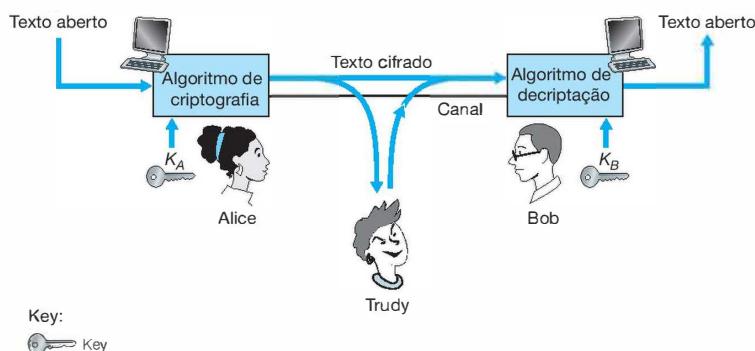
Estabelecida a estrutura, apresentadas algumas das definições mais importantes e justificada a necessidade de segurança na rede, vamos examinar a criptografia mais a fundo. Embora sua utilização para prover confidencialidade seja evidente por si só, veremos em breve que a criptografia também é essencial para prover autenticação do ponto final e integridade de mensagem — o que faz dela uma pedra fundamental da segurança na rede.

8.2 PRINCÍPIOS DE CRIPTOGRAFIA

Embora a criptografia tenha uma longa história que remonta, no mínimo, a Júlio César, técnicas modernas, incluindo muitas das usadas na Internet, são baseadas em progressos feitos nos últimos 30 anos. O livro de Kahn, *The codebreakers* [Kahn, 1967], e o livro de Singh, *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography* [Singh, 1999], nos oferecem um panorama fascinante dessa longa história. Uma discussão completa sobre a criptografia exige um livro inteiro [Kaufman, 1995; Schneier, 1995]; portanto, trataremos apenas de seus aspectos essenciais, em particular do modo como as técnicas criptográficas são postas em prática na Internet. Observamos também que, conquanto nessa seção focalizemos a utilização da criptografia aplicada à confidencialidade, em breve veremos que as técnicas criptográficas estão inextricavelmente entrelaçadas com a autenticação, a integridade de mensagens, o não repúdio e outras.

Técnicas criptográficas permitem que um remetente disfarce os dados de modo que um intruso não consiga obter nenhuma informação dos dados interceptados. O destinatário, é claro, deve estar habilitado a recuperar os

dados originais a partir dos dados disfarçados. A Figura 8.2 apresenta alguns dos componentes mais importantes da terminologia usada em criptografia.

Suponha agora que Alice queira enviar uma mensagem a Bob. A mensagem de Alice em sua forma original (por exemplo, "`Bob, I love you. Alice`") é conhecida como **texto aberto** ou **texto claro**. Alice criptografa sua mensagem em texto aberto usando um **algoritmo de criptografia**, de modo que a mensagem criptografada, conhecida como **texto cifrado**, pareça ininteligível para qualquer intruso. O interessante é que em muitos sistemas criptográficos modernos, incluindo os usados na Internet, a técnica de codificação é *conhecida* — publicada, padronizada e disponível para qualquer um (por exemplo, [RFC 1321; RFC 3447; RFC 2420; NIST, 2001]), mesmo para um potencial intruso! Evidentemente, se todos conhecem o método para codificar dados, então deve haver alguma informação secreta que impede que um intruso decifre os dados transmitidos. É aqui que entra a chave.

Na Figura 8.2, Alice fornece uma **chave**, K_A, uma cadeia de números ou de caracteres, como entrada para o algoritmo de criptografia. O algoritmo pega essa chave e o texto aberto da mensagem, m, como entrada e produz texto cifrado como saída. A notação $K_A(m)$ refere-se à forma do texto cifrado (criptografado usando a chave K_A) da mensagem em texto aberto, m. O algoritmo criptográfico propriamente dito, que usa a chave K_A, ficará evidente do próprio contexto. De maneira semelhante, Bob fornecerá uma chave, K_B, ao **algoritmo de decriptação**, que pega o texto cifrado e a chave de Bob como entrada e produz o texto aberto original como saída. Isto é, se Bob receber uma mensagem criptografada $K_A(m)$, ele a decriptará calculando $K_B(K_A(m)) = m$. Em **sistemas de chaves simétricas**, as chaves de Bob e de Alice são idênticas e secretas. Em **sistemas de chaves públicas**, é usado um par de chaves. Uma delas é conhecida por Bob e por Alice (na verdade, é conhecida pelo mundo inteiro). A outra chave é conhecida apenas por Bob ou por Alice (mas não por ambos). Nas duas subseções seguintes, examinaremos com mais detalhes sistemas de chaves simétricas e de chaves públicas.

8.2.1 Criptografia de chaves simétricas

Todos os algoritmos criptográficos envolvem a substituição de um dado por outro, como tomar um trecho de um texto aberto e então, calculando e substituindo esse texto por outro cifrado apropriado, criar uma mensagem cifrada. Antes de estudar um sistema criptográfico moderno baseado em chaves, vamos abordar um algoritmo de chaves simétricas muito antigo, muito simples, atribuído a Júlio César e conhecido como **cifra de César** (uma cifra é um método para codificar dados).

A cifra de César funciona tomando cada letra da mensagem do texto aberto e substituindo-a pela k-ésima letra sucessiva do alfabeto (permitindo a rotatividade do alfabeto, isto é, a letra "*z*" seria seguida novamente da letra "*a*"). Por exemplo, se $k = 3$, então a letra "*a*" do texto aberto fica sendo "*d*" no texto cifrado; "*b*" no texto aberto

FIGURA 8.2 COMPONENTES CRIPTOGRÁFICOS

se transforma em "*e*" no texto cifrado, e assim por diante. Nesse caso, o valor de *k* serve de chave. Por exemplo, a mensagem "`bob, i love you. alice`" se torna "`ere, l oryh brx. dolfh`" em texto cifrado. Embora o texto cifrado na verdade pareça não ter nexo, você não levaria muito tempo para quebrar o código se soubesse que foi usada a cifra de César, pois há somente 25 valores possíveis para as chaves.

Um aprimoramento da cifra de César é a **cifra monoalfabética**, que também substitui uma letra do alfabeto por outra. Contudo, em vez de fazer isso seguindo um padrão regular (por exemplo, substituição por um deslocamento de *k* para todas as letras), qualquer letra pode ser substituída por qualquer outra, contanto que cada letra tenha uma única substituta e vice-versa. A regra de substituição apresentada na Figura 8.3 mostra uma regra possível para codificar textos abertos.

A mensagem do texto aberto "`bob, I love you. alice`" se torna "`nkn, s gktc wky. mgsbc.`" Assim, como aconteceu no caso da cifra de César, o texto parece sem nexo. A cifra monoalfabética também parece ser melhor que a cifra de César, pois há 26! (da ordem de 10^{26}) possíveis pares de letras, em vez de 25 pares possíveis. Uma técnica de força bruta que experimentasse todos os 10^{26} pares possíveis demandaria um esforço grande demais e impediria que esse fosse um método viável para quebrar o algoritmo criptográfico e decodificar a mensagem. Contudo, pela análise estatística da linguagem do texto aberto, por exemplo, e sabendo que as letras "e" e 't" são as mais frequentes em textos em inglês (13% e 9% das ocorrências de letras, respectivamente) e sabendo também que determinados grupos de duas e de três letras aparecem com bastante frequência em inglês (por exemplo, "in", "it", "the", "ion", "ing", e assim por diante), torna-se relativamente fácil quebrar esse código. Se o intruso tiver algum conhecimento sobre o possível texto da mensagem, então ficará mais fácil ainda quebrar o código. Por exemplo, se a intrusa Trudy for a esposa de Bob e suspeitar que ele está tendo um caso com Alice, ela poderá facilmente imaginar que os nomes "bob" e "alice" apareçam no texto. Se Trudy tivesse certeza de que esses dois nomes aparecem no texto cifrado e tivesse uma cópia do texto cifrado da mensagem do exemplo, então ela poderia determinar imediatamente sete dos 26 pares de letras, o que resultaria em 109 possibilidades a menos para verificar pelo método da força bruta. Na verdade, se Trudy suspeitasse que Bob estava tendo um caso, ela poderia muito bem esperar encontrar algumas outras palavras preferenciais no texto também.

Considerando como seria fácil para Trudy quebrar o código criptográfico de Bob e Alice, podemos distinguir três cenários diferentes, dependendo do tipo de informação que o intruso tem:

- *Ataque exclusivo a texto cifrado*. Em alguns casos, o intruso pode ter acesso somente ao texto cifrado interceptado, sem ter nenhuma informação exata sobre o conteúdo do texto aberto. Já vimos como a análise estatística pode ajudar o **ataque exclusivo ao texto cifrado** em um esquema criptográfico.
- *Ataque com texto aberto conhecido*. Vimos anteriormente que, se Trudy, por alguma razão, tivesse certeza de que "bob" e "alice" apareciam no texto cifrado, ela poderia determinar os pares (texto cifrado, texto aberto) para as letras *a*, *l*, *i*, *c*, *e*, *b* e *o*. Trudy poderia também ser muito sortuda e ter gravado todas as transmissões de texto cifrado e descoberto uma versão decifrada pelo próprio Bob escrita em um pedaço de papel. Quando um intruso conhece alguns dos pares (texto aberto, texto cifrado), referimo-nos a isso como ataque ao esquema criptográfico a partir de **texto aberto conhecido**.
- *Ataque com texto aberto escolhido*. Nesse tipo de ataque, o intruso pode escolher a mensagem em texto aberto e obter seu texto cifrado correspondente. Para os algoritmos criptográficos simples que vimos até aqui, se Trudy conseguisse que Alice enviasse a mensagem "`the quick fox jumps over the lazy brown dog`", ela poderia decifrar completamente o esquema criptográfico. Veremos em breve que, para técnicas de criptografia mais sofisticadas, um ataque com um texto aberto escolhido não significa necessariamente que a técnica criptográfica possa ser decifrada.

FIGURA 8.3 UM CIFRA MONOALFABÉTICA

Letra no texto aberto:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Letra no texto cifrado:	m	n	b	v	c	x	z	a	s	d	f	g	h	j	k	l	p	o	i	u	y	t	r	e	w	q

Quinhentos anos atrás foram inventadas técnicas que aprimoravam a cifra monoalfabética, conhecidas como cifras polialfabéticas. A ideia subjacente à **criptografia polialfabética** é usar várias cifras monoalfabéticas com uma cifra monoalfabética específica para codificar uma letra em uma posição específica no texto aberto da mensagem. Assim, a mesma letra, quando aparece em posições diferentes no texto aberto da mensagem, pode ser codificada de maneira diferente. Um exemplo de esquema criptográfico polialfabético é mostrado na Figura 8.4, na qual há duas cifras de César (com $k = 5$ e $k = 19$), que aparecem nas linhas da figura. Podemos optar pelo uso dessas duas cifras de César, C1 e C2, seguindo o modelo de repetição C1, C2, C2, C1, C2. Isto é, a primeira letra do texto deve ser cifrada usando-se C1, a segunda e a terceira, C2, a quarta, C1 e a quinta, C2. O modelo, então, se repete, com a sexta letra sendo cifrada usando-se C1; a sétima, com C2, e assim por diante. Dessa maneira, a mensagem em texto aberto "`bob, I love you.`" é cifrada como "`ghu, n etox dhz.`". Note que o primeiro "b" da mensagem em texto aberto é cifrado usando-se C1, ao passo que o segundo "b" é cifrado usando-se C2. Nesse exemplo, a "chave" da codificação e da decodificação é o conhecimento das duas cifras de César ($k = 5$, $k = 19$) e do modelo C1, C2, C2, C1, C2.

Cifras de bloco

Vamos agora nos direcionar a tempos modernos e analisar como a criptografia de chave simétrica é feita atualmente. Existem duas classes gerais de técnicas de criptografia simétrica: **cifras de fluxo** e **cifras de bloco**. Na Seção 8.7, examinaremos as cifras de fluxo de forma breve, ao investigarmos a segurança para LANs sem fio. Nesta seção, focaremos em cifras de bloco, que são utilizadas em muitos protocolos seguros da Internet, incluindo PGP (para e-mail seguro), SSL (para conexões TCP seguras) e IPsec (para proteger o transporte da camada de rede).

Na cifra de bloco, a mensagem a ser criptografada é processada em blocos de k bits. Por exemplo, se $k = 64$, então a mensagem é dividida em blocos de 64 bits, e cada bloco é criptografado de maneira independente. Para criptografar um bloco, a cifra utiliza um mapeamento um para um para mapear o bloco de k bits de texto aberto para um bloco de k bits de texto cifrado. Vamos examinar um exemplo. Suponha que $k = 3$, de modo que a cifra de bloco mapeie entradas de 3 bits (texto aberto) para saídas de 3 bits (texto cifrado). Um possível mapeamento é determinado na Tabela 8.1. Observe que esse é um mapeamento um para um; ou seja, há uma saída diferente para cada entrada. Essa cifra de bloco divide a mensagem em blocos de até 3 bits e criptografa cada bloco de acordo com o mapeamento acima. Você deve verificar que a mensagem 010110001111 é criptografada para 101000111001.

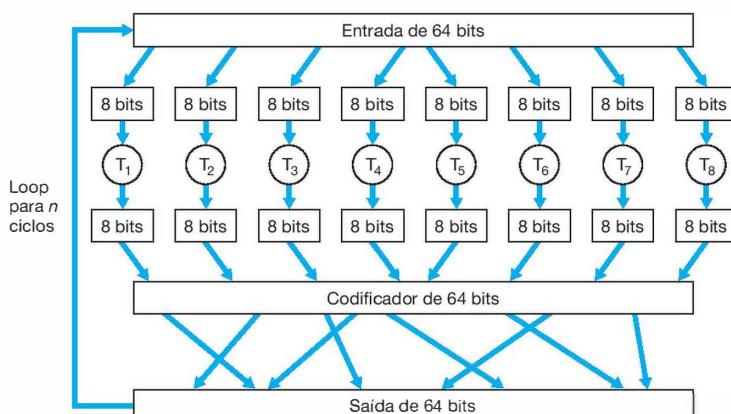
Ainda no exemplo do bloco de 3 bits, observe que o mapeamento na Tabela 8.1 é um de muitos possíveis mapeamentos. Quantos deles existem? Para responder a essa questão, observe que um mapeamento nada mais é do que uma permutação de todas as possíveis entradas. Há 2^3 (= 8) possíveis entradas (relacionadas nas colunas de entrada). Elas podem ser permutadas em 8! = 40.320 formas diferentes. Uma vez que cada permutação especifique

FIGURA 8.4 UMA CIFRA POLIALFABÉTICA QUE UTILIZA DUAS CIFRAS DE CÉSAR

Letra do texto aberto:	a b c d e f g h i j k l m n o p q r s t u v w x y z
$C_1(k = 5)$:	f g h i j k l m n o p q r s t u v w x y z a b c d e
$C_2(k = 19)$:	t u v w x y z a b c d e f g h i j k l m n o p q r s

TABELA 8.1 UMA CIFRA DE BLOCO DE 3 BITS ESPECÍFICA

Entrada	Saída	Entrada	Saída
000	110	100	011
001	111	101	010
010	101	110	000
011	100	111	001

um mapeamento, há 40.320 mapeamentos possíveis. Podemos ver cada mapeamento como uma chave — se Alice e Bob sabem o mapeamento (a chave), conseguem criptografar e decodificar as mensagens enviadas entre eles.

O ataque de força bruta para essa cifra é tentar decodificar o *texto cifrado* usando todos os mapeamentos. Com apenas 40.320 mapeamentos (quando $k = 3$), isso pode ser rapidamente realizado em um computador de mesa. Para impedir os ataques de força bruta, as cifras de bloco normalmente usam blocos muito maiores, consistindo em $k = 64$ bits ou ainda maior. Observe que o número de mapeamentos possíveis para uma cifra de bloco k geral é $2^k!$, o qual é extraordinário para até mesmo valores moderados de k (como $k = 64$).

Embora as cifras de bloco da tabela completa, como descritas, com valores moderados de k possam produzir esquemas robustos de criptografia de chave simétrica, eles infelizmente são difíceis de implementar. Para $k = 64$ e para um determinado mapeamento, Alice e Bob precisariam manter uma tabela com 2^{64} valores de entrada, uma tarefa impraticável. Além disso, se Alice e Bob quiserem trocar chaves, cada um teria de renovar a tabela. Assim, a cifra de bloco da tabela completa, que fornece mapeamentos predeterminados entre todas as entradas e saídas (como no exemplo anterior), está simplesmente fora de cogitação.

Em vez disso, as cifras de bloco costumam utilizar funções que simulam, de maneira aleatória, tabelas permutadas. Um exemplo (adaptado de Kaufman [1995]) de tal função para $k = 64$ bits é mostrado na Figura 8.5. A função primeiro divide um bloco de 64 bits em 8 blocos, cada qual consistindo de 8 bits. Cada bloco de 8 bits é processado por uma tabela de 8 bits para 8 bits, a qual possui um tamanho controlável. Por exemplo, o primeiro bloco é processado pela tabela denotada por T_1. Em seguida, os oito blocos de saída são reunidos em um bloco de 64 bits. As posições dos 64 bits no bloco são, então, permutadas para produzir uma saída de 64 bits. Essa saída é devolvida à entrada de 64 bits, onde se inicia outro ciclo. Após n ciclos, a função apresenta um bloco de 64 bits de texto cifrado. O objetivo de cada ciclo é fazer cada bit de entrada afetar a maioria (se não todos) dos bits finais de saída. (Se somente um ciclo fosse usado, um determinado bit de entrada afetaria somente 8 dos 64 bits de saída.) A chave para esse algoritmo das cifras de bloco seria as oito tabelas de permutação (admitindo que a função de permutação seja publicamente conhecida).

Hoje, existem diversas cifras de bloco conhecidas, incluindo DES (abreviação de *Data Encryption Standard* [Padrão de Criptografia de Dados]), 3DES e AES (abreviação de *Advanced Encryption Standard* [Padrão de Criptografia Avançada]). Cada padrão utiliza funções em vez de tabelas predeterminadas, segundo a Figura 8.5 (embora mais complexa e específica para cada cifra). Cada um desses algoritmos também utiliza uma cadeia de bits para chave. Por exemplo, o DES usa blocos de 64 bits com uma chave de 56 bits. O AES usa blocos de 128 bits e pode operar com chaves de 128, 192 e 256 bits de comprimento. A chave de um algoritmo determina os mapeamentos da "minitabela" e permutações dentro do algoritmo. O ataque de força bruta para cada uma dessas

FIGURA 8.5 EXEMPLO DE UMA CIFRA DE BLOCO

cifras é percorrer todas as chaves, aplicando o algoritmo de decriptografia com cada chave. Observe que com o comprimento de chave n, há 2^n chaves possíveis. NIST [2001] estima que uma máquina que pudesse decifrar um DES de 56 bits em um segundo (ou seja, testar 2^{56} chaves em um segundo) levaria, mais ou menos, 149 trilhões de anos para decifrar uma chave AES de 128 bits.

Encadeamento de blocos de cifras

Em aplicações de redes de computadores, em geral precisamos criptografar mensagens longas (ou fluxos de dados longos). Se aplicarmos uma cifra de bloco, como descrita, apenas partindo a mensagem em blocos de k bits e criptografando de modo independente cada bloco, um problema sutil mas importante ocorrerá. Para entendê-lo, note que dois ou mais blocos de texto aberto podem ser idênticos. Por exemplo, o texto aberto em dois ou mais blocos poderia ser "HTTP/1.1". Em relação a esses blocos idênticos, uma cifra de bloco produziria, é claro, o mesmo texto cifrado. Um atacante poderia talvez adivinhar o texto aberto ao ver blocos de texto cifrado idênticos e ser capaz de decodificar a mensagem inteira identificando os blocos de texto cifrado idênticos e usando o que sabe sobre a estrutura do protocolo subjacente [Kaufman, 1995].

Para abordar esse problema, podemos associar um pouco de aleatoriedade ao texto cifrado para que blocos de texto aberto idênticos produzam blocos de texto cifrado diferentes. Para explicar essa ideia, $m(i)$ representará o i-ésimo bloco de texto aberto, $c(i)$ representará o i-ésimo bloco de texto cifrado, e $a \oplus b$ representará o ou-exclusivo (XOR) das duas cadeias de bits, a e b. (Lembre-se de que $0 \oplus 0 = 1 \oplus 1 = 0$ e $0 \oplus 1 = 1 \oplus 0 = 1$, e o XOR das duas cadeias de bits é feito em uma base de bit por bit. Então, por exemplo, $10101010 \oplus 11110000 = 01011010$.) Ademais, denote o algoritmo de criptografia da cifra de bloco com a chave S como K_S. Eis a ideia básica. O emissor cria um número aleatório $r(i)$ de k bits para o i-ésimo bloco e calcula $c(i) = K_S\ (m(i) \oplus r(i))$. Observe que um novo número aleatório de k bits é escolhido para cada bloco. O emissor envia, então, $c(1)$, $r(1)$, $c(2)$, $r(2)$, $c(3)$, $r(3)$ etc. Visto que o receptor recebe $c(i)$ e $r(i)$, ele pode recuperar cada bloco de texto aberto computando $m(i) = K_S\ (c(i)) \oplus r(i)$. É importante observar que, embora $r(i)$ seja enviado de modo inocente e, portanto, pode ser analisada por Trudy, ela não consegue obter blocos de texto aberto $m(i)$, pois não conhece a chave K_S. Observe também que se dois blocos de texto aberto $m(i)$ e $m\ (j)$ são iguais, os blocos de texto cifrado correspondentes $c(i)$ e $c(j)$ serão diferentes (contanto que os números $r(i)$ e $r(j)$ aleatórios sejam diferentes, o que acontece com alta probabilidade).

Como um exemplo, considere a cifra de bloco de 3 bits na Tabela 8.1. Suponha que o texto aberto seja 010010010. Se Alice criptografá-lo diretamente, sem incluir a aleatoriedade, o texto cifrado resultante se torna 101101101. Se Trudy analisar esse texto cifrado, como cada uma das três cifras de blocos é igual, ela pode pensar que cada um dos blocos de texto aberto são iguais. Agora suponha que em vez disso Alice cria blocos aleatórios $r(1) = 001$, $r(2) = 111$, e $r(3) = 100$ e use a técnica anterior para criar o texto cifrado $c(1) = 100$, $c(2) = 010$ e $c(3) = 000$. Observe que os três blocos de texto cifrado são diferentes mesmo se os blocos de texto aberto forem iguais. Alice então envia $c(1)$, $r(1)$, $c(2)$ e $r(2)$. Você deve verificar que Bob pode obter o texto aberto original usando a chave K_S compartilhada.

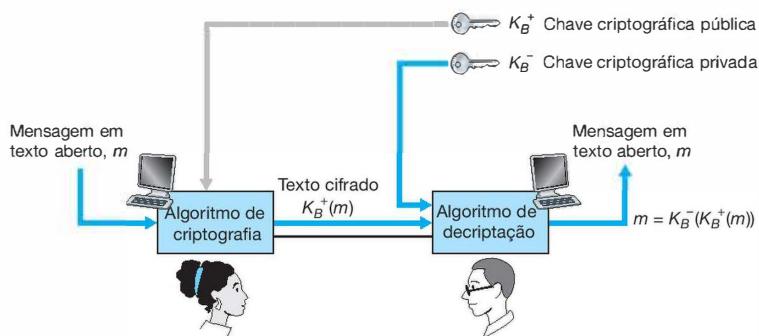
O leitor atento observará que introduzir a aleatoriedade resolve o problema, mas cria outro: ou seja, Alice deve transmitir duas vezes mais bits que antes. De fato, para cada bit de cifra, ela deve agora enviar um bit aleatório, dobrando a largura de banda exigida. Para obtermos o melhor dos dois mundos, as cifras de bloco em geral usam uma técnica chamada **Encadeamento do Bloco de Cifra** (CBC — *Cipher Block Chaining*). A ideia básica é enviar somente *um valor aleatório junto com a primeira mensagem e, então, fazer o emissor e o receptor usarem blocos codificados em vez do número aleatório subsequente.* O CBC opera da seguinte forma:

1. Antes de criptografar a mensagem (ou o fluxo de dados), o emissor cria uma cadeia de k bits, chamada **Vetor de Inicialização (IV)**. Indique esse vetor de inicialização por $c(0)$. O emissor envia o IV ao receptor *em texto aberto*.
2. Em relação ao primeiro bloco, o emissor calcula $m(1) \oplus c(0)$, ou seja, calcula o ou-exclusivo do primeiro bloco de texto aberto com o IV. Ele então codifica o resultado através do algoritmo de cifra de bloco para

obter o bloco de texto cifrado correspondente; isto é, $c(1) = K_S(m(1) \oplus c(0))$. O emissor envia o bloco criptografado $c(1)$ ao receptor.

3. Para o i-ésimo bloco, o emissor cria o i-ésimo bloco de texto cifrado de $c(i) = K_S(m(i) \oplus c(i-1))$.

Vamos agora examinar algumas das consequências dessa abordagem. Primeiro, o receptor ainda será capaz de recuperar a mensagem original. De fato, quando o receptor recebe $c(i)$, ele o decodifica com K_S para obter $s(i) = m(i) \oplus c(i-1)$; uma vez que o receptor também conhece $c(i-1)$, ele então obtém o bloco de texto aberto de $m(i) = s(i) \oplus c(i-1)$. Segundo, mesmo se dois blocos de texto aberto forem idênticos, os texto cifrados correspondentes (quase sempre) serão diferentes. Terceiro, embora o emissor envie o IV aberto, um invasor ainda não será capaz de decodificar os blocos de texto cifrado, visto que o invasor não conhece a chave secreta, S. Por fim, o emissor somente envia um bloco auxiliar (o IV), fazendo aumentar, de forma insignificante, o uso da largura de banda para longas mensagens (consistindo de centenas de blocos).

Como um exemplo, vamos determinar o texto cifrado para uma cifra de bloco de 3 bits na Tabela 8.1 com texto aberto 010010010 e IV = $c(0)$ = 001. O emissor primeiro usa o IV para calcular $c(1) = K_S(m(1) \oplus c(0)) = 100$. O emissor calcula, então, $c(2) = K_S(m(2) \oplus c(1)) = K_S(010 \oplus 100) = 000$, e $c(3) = K_S(m(3) \oplus c(2)) = K_S(010 \oplus 000) = 101$. O leitor deve verificar que o receptor, conhecendo o IV e K_S, pode recuperar o texto aberto original.

O CBC possui uma consequência importante ao projetar protocolos de rede seguros: precisaremos fornecer um mecanismo dentro do protocolo para distribuir o IV do emissor ao receptor. Veremos como isso é feito para vários protocolos mais adiante, neste capítulo.

8.2.2 Criptografia de chave pública

Por mais de dois mil anos (desde a época da cifra de César até a década de 1970), a comunicação cifrada exigia que as duas partes comunicantes compartilhassem um segredo — a chave simétrica usada para cifrar e decifrar. Uma dificuldade dessa abordagem é que as duas partes têm de concordar, de alguma maneira, com a chave compartilhada, mas, para fazê-lo, é preciso comunicação (presumivelmente *segura*)! Talvez as partes pudessem se encontrar antes, escolher a chave (por exemplo, dois dos centuriões de César poderiam se encontrar nos banhos romanos) e, mais tarde, se comunicar de modo cifrado. Em um mundo em rede, contudo, o mais provável é que as partes comunicantes nunca possam se encontrar e jamais consigam conversar a não ser pela rede. É possível que elas se comuniquem por criptografia sem compartilhar uma chave comum secreta conhecida com antecedência? Em 1976, Diffie e Hellman [Diffie, 1976] apresentaram um algoritmo (conhecido como Troca de Chaves Diffie Hellman — *Diffie Hellman Key Exchange*) que faz exatamente isso — uma abordagem da comunicação segura radicalmente diferente e de uma elegância maravilhosa que levou ao desenvolvimento dos atuais sistemas de criptografia de chaves públicas. Veremos em breve que os sistemas de criptografia de chaves públicas também têm diversas propriedades maravilhosas que os tornam úteis não só para criptografia, mas também para autenticação e assinaturas digitais. É interessante que, há pouco, veio à luz que ideias semelhantes às de [Diffie, 1976] e às da [RSA, 1978] foram desenvolvidas independentemente no início da década de 1970 em uma série de relatórios secretos escritos por pesquisadores do Grupo de Segurança para Comunicação e Eletrônica (Communications-Electronics Security Group) do Reino Unido [Ellis, 1987]. Como acontece com frequência, grandes ideias podem surgir de modo independente em diversos lugares; felizmente, os progressos da criptografia de chaves públicas ocorreram não apenas no âmbito privado, mas também no público.

A utilização de criptografia de chaves públicas, como conceito, é bastante simples. Suponha que Alice queira se comunicar com Bob. Como ilustra a Figura 8.6, em vez de Bob e Alice compartilharem uma única chave secreta (como no caso dos sistemas de chaves simétricas), Bob (o destinatário das mensagens de Alice) tem duas chaves — uma **chave pública**, que está à disposição do *mundo todo* (inclusive de Trudy, a intrusa), e uma **chave privada**, que apenas ele (Bob) conhece. Usaremos a notação K_B^+ e K_B^- para nos referirmos às chaves pública e privada de Bob, respectivamente. Para se comunicar com Bob, Alice busca primeiro a chave pública de Bob. Em seguida, ela criptografa sua mensagem, m, usando a chave pública de Bob e um algoritmo criptográfico conhecido (por exemplo, padronizado), isto é, Alice calcula $K_B^+(m)$. Bob recebe a mensagem criptografada de Alice e usa sua chave privada e um algoritmo de decriptação conhecido (por exemplo, padronizado) para decifrar a mensagem de Alice,

FIGURA 8.6 CRIPTOGRAFIA DE CHAVES PÚBLICAS

isto é, Bob calcula $K_B^-(K_B^+(m))$. Veremos, mais adiante, que há algoritmos e técnicas de criptografia/decriptação para escolher chaves públicas e privadas de modo que $K_B^-(K_B^+(m)) = m$; isto é, aplicando a chave pública de Bob, K_B^+, à mensagem m, para obter $K_B^+(m)$, e então aplicando a chave privada de Bob, K_B^-, à versão criptografada de m, isto é, calculando $K_B^-(K_B^+(m))$, obtemos m novamente. Esse resultado é notável! Dessa maneira, Alice pode utilizar a chave de Bob disponível publicamente para enviar uma mensagem secreta a Bob sem que nenhum deles tenha de permutar nenhuma chave secreta! Veremos em breve que nós podemos permutar a chave pública e a chave privada de criptografia e obter o mesmo resultado notável, isto é, $K_B^-(K_B^+(m)) = K_B^+(K_B^-(m)) = m$.

A utilização de criptografia de chave pública é, portanto, conceitualmente simples. Mas há duas preocupações que podem nos ocorrer de imediato. A primeira é que, embora um intruso que intercepte a mensagem cifrada de Alice veja apenas dados ininteligíveis, ele conhece tanto a chave (a chave pública de Bob, disponível a todos) quanto o algoritmo que Alice usou para a criptografia. Assim, Trudy pode montar um ataque com texto aberto escolhido utilizando o algoritmo criptográfico padronizado conhecido e a chave criptográfica de acesso público de Bob para codificar a mensagem que quiser! Ela pode até tentar, por exemplo, codificar mensagens, ou partes delas, que suspeita que Alice poderia enviar. Fica claro que, para a criptografia de chave pública funcionar, a escolha de chaves e de códigos de criptografia/decriptação deve ser feita de tal maneira que seja impossível (ou, ao menos, tão difícil que se torne quase impossível) para um intruso determinar a chave privada de Bob ou conseguir decifrar ou adivinhar a mensagem de Alice a Bob. A segunda preocupação é que, como a chave criptográfica de Bob é pública, qualquer um pode enviar uma mensagem cifrada a Bob, incluindo Alice ou alguém *afirmando ser* Alice. No caso de uma única chave secreta compartilhada, o fato de o remetente conhecer a chave secreta identifica implicitamente o remetente para o destinatário. No caso da criptografia de chave pública, contudo, isso não acontece, já que qualquer um pode enviar uma mensagem cifrada a Bob usando a chave dele, que está publicamente disponível a todos. É preciso uma assinatura digital, um tópico que estudaremos na Seção 8.3, para vincular um remetente a uma mensagem.

RSA

Embora existam muitos algoritmos e chaves que tratam dessas preocupações, o **algoritmo RSA** (cujo nome se deve a seus inventores, Ron Rivest, Adi Shamir e Leonard Adleman) tornou-se quase um sinônimo de criptografia de chave pública. Vamos, primeiro, ver como o RSA funciona e, depois, examinar por que ele funciona.

O RSA faz uso extensivo das operações aritméticas usando a aritmética de módulo-n. Vamos revisar de maneira breve a aritmética modular. Lembre-se de que x mod n simplesmente significa o resto de x quando dividido por n, de modo que, por exemplo, 19 mod 5 = 4. Na aritmética modular, uma pessoa executa as operações comuns de adição, multiplicação e exponenciação. Entretanto, o resultado de cada operação é substituído pelo resto inteiro que sobra quando o resultado é dividido por n. A adição e a multiplicação com a aritmética modular são facilitadas com as seguintes propriedades úteis:

$$[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$$
$$[(a \bmod n) - (b \bmod n)] \bmod n = (a - b) \bmod n$$
$$[(a \bmod n) \bullet (b \bmod n)] \bmod n = (a \bullet b) \bmod n$$

Segue da terceira propriedade que $(a \bmod n)^d \bmod n = a^d \bmod n$, uma identidade que em breve acharemos muito útil.

Agora suponha que Alice queira enviar a Bob uma mensagem criptografada por meio do RSA, conforme ilustrado na Figura 8.6. Em nossa discussão sobre o RSA, vamos sempre manter em mente que uma mensagem não é nada mais do que um padrão de bits, e cada padrão de bits pode ser representado unicamente por um número inteiro (junto com o comprimento do padrão de bits). Por exemplo, suponha que uma mensagem tenha o padrão de bits 1001; essa mensagem pode ser representada pelo número inteiro decimal 9. Assim, criptografar uma mensagem com RSA é equivalente a criptografar um número inteiro que representa a mensagem.

Existem dois componentes inter-relacionados do RSA:

- A escolha da chave pública e da chave privada.
- O algoritmo de criptografia/decriptação.

Para escolher as chaves pública e privada no RSA, Bob deve executar as seguintes etapas:

1. Escolher dois números primos grandes, p e q. Que ordem de grandeza devem ter p e q? Quanto maiores os valores, mais difícil será quebrar o RSA, mas mais tempo se levará para realizar a codificação e a decodificação. O RSA Laboratories recomenda que o produto de p e q seja da ordem de 1.024 bits. Para uma discussão sobre como achar números primos grandes, consulte Caldwell [2012].
2. Calcular $n = pq$ e $z = (p - 1)(q - 1)$.
3. Escolher um número e menor do que n que não tenha fatores comuns (exceto o 1) com z. (Nesse caso, dizemos que e e z são números primos entre si.) A letra "e" é usada já que esse valor será utilizado na criptografia ("*encryption*", em inglês).
4. Achar um número d, tal que $ed - 1$ seja divisível exatamente (isto é, não haja resto na divisão) por z. A letra "d" é usada porque seu valor será utilizado na decriptação. Em outras palavras, dado e, escolhemos d tal que

$$ed \bmod z = 1$$

5. A chave pública que Bob põe à disposição de todos, K_B^+, é o par de números (n, e); sua chave privada, K_B^-, é o par de números (n, d).

A criptografia feita por Alice e a decriptação feita por Bob acontecem da seguinte forma:

- Suponha que Alice queira enviar a Bob um padrão de bits, ou número m, tal que $m < n$. Para codificar, Alice calcula a potência m^e e, então, determina o resto inteiro da divisão de m^e por n. Assim, o valor cifrado, c, da mensagem em texto aberto de Alice, m, é:

$$c = m^e \bmod n$$

O padrão de bits correspondente a esse texto cifrado c é enviado a Bob.
- Para decifrar a mensagem em texto cifrado recebida, c, Bob calcula

$$m = c^d \bmod n$$

que exige o uso de sua chave secreta (n, d).

Como exemplo simples de RSA, suponha que Bob escolha $p = 5$ e $q = 7$. (Admitimos que esses valores são muito pequenos para ser seguros.) Então, $n = 35$ e $z = 24$. Bob escolhe $e = 5$, já que 5 e 24 não têm fatores comuns. Por fim, ele escolhe $d = 29$, já que $5 \cdot 29 - 1$ (isto é, $ed - 1$) é divisível exatamente por 24. Ele divulga os dois valores, $n = 35$ e $e = 5$, e mantém em segredo o valor $d = 29$. Observando esses dois valores públicos, suponha que Alice queira agora enviar as letras *l*, *o*, *v* e *e* a Bob. Interpretando cada letra como um número entre 1 e 26 (com *a* sendo 1 e *z* sendo 26), Alice e Bob realizam a criptografia e a decriptação mostradas nas tabelas 8.2 e 8.3,

respectivamente. Observe que neste exemplo consideramos cada uma das quatro letras como uma mensagem distinta. Um exemplo mais realista seria converter as quatro letras em suas representações ASCII de 8 bits e então codificar o número inteiro correspondente ao padrão de bits de 32 bits resultante. (Esse exemplo realista cria números muito longos para publicar neste livro!)

Dado que o exemplo fictício das tabelas 8.2 e 8.3 já produziu alguns números extremamente grandes e visto que sabemos, porque vimos antes, que p e q devem ter, cada um, algumas centenas de bits de comprimento, várias questões práticas nos vêm à mente no caso do RSA. Como escolher números primos grandes? Como escolher e e d? Como calcular exponenciais de números grandes? A discussão desses assuntos está além do escopo deste livro; consulte Kaufman [1995] e as referências ali citadas para obter mais detalhes.

TABELA 8.2 CRIPTOGRAFIA RSA PARA ALICE, E = 5, N = 35

Letra do texto aberto	m: representação numérica	m^e	Texto cifrado $c = m^e \bmod n$
l	12	248832	17
o	15	759375	15
v	22	5153632	22
e	5	3125	10

TABELA 8.3 DECRIPTAÇÃO RSA PARA BOB, D = 29, N = 35

Texto cifrado c	c^d	$m = c^d \bmod n$	Letra do texto aberto
17	481968572106750915091411825223071697	12	l
15	12783403940394885893911123275756835937 5	15	o
22	851643319086537701956194499721106030592	22	v
10	100000000000000000000000000000	5	e

Chaves de sessão

Observamos aqui que a exponenciação exigida pelo RSA é um processo que consome tempo considerável. O DES, ao contrário, é, no mínimo, cem vezes mais veloz em software e entre mil e dez mil vezes mais veloz em hardware [RSA Fast, 2012]. Como resultado, o RSA é frequentemente usado na prática em combinação com a criptografia de chave simétrica. Por exemplo, se Alice quer enviar a Bob uma grande quantidade de dados cifrados a alta velocidade, ela pode fazer o seguinte. Primeiro escolhe uma chave que será utilizada para codificar os dados em si; essa chave às vezes é denominada **chave de sessão**, representada por K_S. Alice deve informar a Bob essa chave de sessão, já que essa é a chave simétrica compartilhada que eles usarão com uma cifra de chave simétrica (por exemplo, DES ou AES). Alice criptografa o valor da chave de sessão usando a chave pública RSA de Bob, isto é, ela processa $c = (K_S)^e \bmod n$. Bob recebe a chave de sessão codificada RSA, c, e a decifra para obter a chave de sessão K_S. Ele agora conhece a chave que Alice usará para transferir dados cifrados.

Por que o RSA funciona?

A criptografia/decriptação do RSA parece mágica. Por que será que, aplicando o algoritmo de criptografia e, em seguida, o de decriptação, podemos recuperar a mensagem original? Para entender por que o RSA funciona, tome de novo $n = pq$, onde p e q são os números primos grandes usados no algoritmo RSA.

Lembre-se de que, na criptografia RSA, uma mensagem (representada exclusivamente por um número inteiro) m é elevada primeiro à potência e usando-se aritmética de módulo n, ou seja,

$$c = m^e \bmod n$$

A decriptação é feita elevando-se esse valor à potência d, novamente usando a aritmética de módulo n. O resultado de uma etapa de criptografia, seguida de uma etapa de decriptação, é então $(m^e \bmod n)^d \bmod n$. Vamos ver agora o que podemos dizer sobre essa quantidade. Como mencionado antes, uma propriedade importante da aritmética modular é $(a \bmod n)^d \bmod n = a^d \bmod n$ para quaisquer valores a, n e d. Então, usando $a = m^e$ nesta propriedade, temos:

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$

Portanto, falta mostrar que $m^{ed} \bmod n = m$. Embora estejamos tentando eliminar um pouco da mágica do modo de funcionamento do RSA, para explicá-lo, precisaremos usar, aqui, outro resultado bastante mágico da teoria dos números. Especificamente, precisamos do resultado que diga que, se p e q forem primos, $n = pq$, e $z = (p - 1)(q - 1)$, então $x^y \bmod n$ será o mesmo que $x^{(y \bmod z)} \bmod n$ [Kaufman, 1995]. Aplicando esse resultado com $x = m$ e $y = ed$, temos

$$m^{ed} \bmod n = m^{(ed \bmod z)} \bmod n$$

Mas lembre-se de que escolhemos e e d tais que $ed \bmod z = 1$. Isso nos dá

$$m^{ed} \bmod n = m^1 \bmod n = m$$

que é exatamente o resultado que esperávamos! Efetuando primeiro a exponenciação da potência e (isto é, criptografando) e depois a exponenciação da potência d (isto é, decriptando), obtemos o valor original m. E mais notável ainda é o fato de que, se elevarmos primeiro à potência d e, em seguida, à potência e, isto é, se invertermos a ordem da criptografia e da decriptação, realizando primeiro a operação de decriptação e, em seguida, aplicando a de criptografia — também obteremos o valor original m. Esse resultado extraordinário resulta imediatamente da aritmética modular:

$$(m^d \bmod n)^e \bmod n = m^{de} \bmod n = m^{ed} \bmod n = (m^e \bmod n)^d \bmod n$$

A segurança do RSA reside no fato de que não se conhecem algoritmos para fatorar rapidamente um número, nesse caso, o valor público n, em números primos p e q. Se alguém conhecesse os números p e q, então, dado o valor público e, poderia com facilidade processar a chave secreta d. Por outro lado, não se sabe se *existem* ou não algoritmos rápidos para fatorar um número e, nesse sentido, a segurança do RSA não é garantida.

Outro conhecido algoritmo de criptografia de chave pública é o Diffie-Hellman, que será explorado resumidamente nos Problemas. O Diffie-Hellman não é tão versátil quanto o RSA, pois não pode ser usado para cifrar mensagens de comprimento arbitrário; ele pode ser usado, entretanto, para determinar uma chave de sessão simétrica, que, por sua vez, é utilizada para codificar mensagens.

8.3 INTEGRIDADE DE MENSAGEM E ASSINATURAS DIGITAIS

Na seção anterior, vimos como a criptografia pode ser usada para oferecer sigilo a duas entidades em comunicação. Nesta seção, nos voltamos ao assunto igualmente importante da criptografia que é prover **integridade da mensagem** (também conhecida como autenticação da mensagem). Além da integridade da mensagem, discutiremos dois assuntos parecidos nesta seção: assinaturas digitais e autenticação do ponto final.

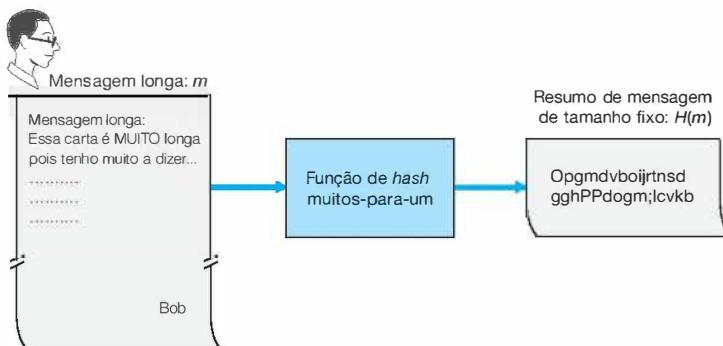
Definimos o problema de integridade da mensagem usando, mais uma vez, Alice e Bob. Suponha que Bob receba uma mensagem (que pode ser cifrada ou estar em texto aberto) acreditando que tenha sido enviada por Alice. Para autenticar a mensagem, Bob precisa verificar se:

1. A mensagem foi, realmente, enviada por Alice.
2. A mensagem não foi alterada em seu caminho para Bob.

Veremos, nas seções 8.4 a 8.7, que esse problema de integridade da mensagem é uma preocupação importante em todos os protocolos de rede seguros.

Como um exemplo específico, considere uma rede de computadores que está utilizando um algoritmo de roteamento de estado de enlace (como OSPF) para determinar rotas entre cada dupla de roteadores na rede (veja Capítulo 4). Em um algoritmo de estado de enlace, cada roteador precisa transmitir uma mensagem de estado de enlace a todos os outros roteadores na rede. Uma mensagem de estado de enlace do roteador inclui uma relação de seus vizinhos diretamente conectados e os custos diretos a eles. Uma vez que o roteador recebe mensagens de estado de enlace de todos os outros roteadores, ele pode criar um mapa completo da rede, executar seu algoritmo de roteamento de menor custo e configurar sua tabela de repasse. Um ataque relativamente fácil no algoritmo de roteamento é Trudy distribuir mensagens de estado de enlace falsas com informações incorretas sobre o estado de enlace. Assim, a necessidade de integridade da mensagem — quando o roteador B recebe uma mensagem de estado de enlace do roteador A, deve verificar que o roteador A na verdade criou a mensagem e que ninguém a alterou em trânsito.

Nesta seção descrevemos uma técnica conhecida sobre integridade da mensagem usada por muitos protocolos de rede seguros. Mas, antes disso, precisamos abordar outro importante tópico na criptografia — as funções de *hash* criptográficas.

8.3.1 Funções de *hash* criptográficas

Como mostrado na Figura 8.7, a função de *hash* recebe uma entrada, m, e calcula uma cadeia de tamanho fixo $H(m)$ conhecida como *hash*. A soma de verificação da Internet (Capítulo 3) e os CRCs (Capítulo 4) satisfazem essa definição. Uma **função *hash* criptográfica** deve apresentar a seguinte propriedade adicional:

- Em termos de processamento, é impraticável encontrar duas mensagens diferentes x e y tais que $H(x) = H(y)$.

Informalmente, essa propriedade significa que, em termos de processamento, é impraticável que um invasor substitua uma mensagem por outra que está protegida pela função *hash*. Ou seja, se (m, H(m)) é a mensagem e o *hash* da mensagem criada pelo emissor, um invasor não pode forjar os conteúdos de outra mensagem, y, que possui o mesmo valor de *hash* da mensagem original.

É bom nos convencermos de que uma soma de verificação simples, como a soma de verificação da Internet, criaria uma função de *hash* criptográfica fraca. Em vez de processar a aritmética de complemento de 1 (como é feito para a soma de verificação da Internet), vamos efetuar uma soma de verificação tratando cada caractere como um byte e somando os bytes usando porções de 4 bytes por vez. Suponha que Bob deva a Alice 100,99 dólares e lhe envie um vale contendo a cadeia de texto "`IOU100.99BOB`" (`IOU — I Owe You` — Eu lhe devo.) A representação ASCII (em notação hexadecimal) para essas letras é `49, 4F, 55, 31, 30, 30, 2E, 39, 39, 42, 4F, 42.`

FIGURA 8.7 FUNÇÕES DE *HASH*

A Figura 8.8 (parte de cima) mostra que a soma de verificação de 4 bytes para essa mensagem é B2 C1 D2 AC. Uma mensagem ligeiramente diferente (e que sairia muito mais cara para Bob) é mostrada na parte de baixo da Figura 8.8. As mensagens "`IOU100.99BOB`" e "`IOU900.19BOB`" têm *a mesma* soma de verificação. Assim, esse algoritmo de soma de verificação simples viola a exigência anterior. Fornecidos os dados originais, é simples descobrir outro conjunto de dados com a mesma soma de verificação. É claro que, para efeito de segurança, precisaremos de uma função de *hash* muito mais poderosa do que uma soma de verificação.

O algoritmo de *hash* MD5 de Ron Rivest [RFC 1321] é amplamente usado hoje. Ele processa um resumo de mensagem de 128 bits por meio de um processo de quatro etapas, constituído de uma etapa de enchimento (adição de um "um" seguido de "zeros" suficientes, para que o comprimento da mensagem satisfaça determinadas condições), uma etapa de anexação (anexação de uma representação de 64 bits do comprimento da mensagem antes do enchimento), uma etapa de inicialização de um acumulador e uma etapa final iterativa, na qual os blocos de 16 palavras da mensagem são processados (misturados) em quatro rodadas de processamento. Para ver uma descrição do MD5 (incluindo uma implementação em código fonte C), consulte [RFC 1321].

O segundo principal algoritmo de *hash* em uso atualmente é o SHA-1 (*Secure Hash Algorithm* — algoritmo de *hash* seguro) [FIPS, 1995]. Esse algoritmo se baseia em princípios similares aos usados no projeto do MD4 [RFC 1320], o predecessor do MD5. O uso do SHA-1, um padrão federal norte-americano, é exigido sempre que um algoritmo de *hash* criptográfico for necessário para aos federais. Ele produz uma mensagem de resumo de 160 bits. O resultado mais longo torna o SHA-1 mais seguro.

FIGURA 8.8 MENSAGEM INICIAL E MENSAGEM FRAUDULENTA TÊM A MESMA SOMA DE VERIFICAÇÃO!

Mensagem	Representação ASCII	
I O U 1	49 4F 55 31	
0 0 . 9	30 30 2E 39	
9 B O B	39 42 4F 42	
	B2 C1 D2 AC	Verificação

Mensagem	Representação ASCII	
I O U 9	49 4F 55 39	
0 0 . 1	30 30 2E 31	
9 B O B	39 42 4F 42	
	B2 C1 D2 AC	Verificação

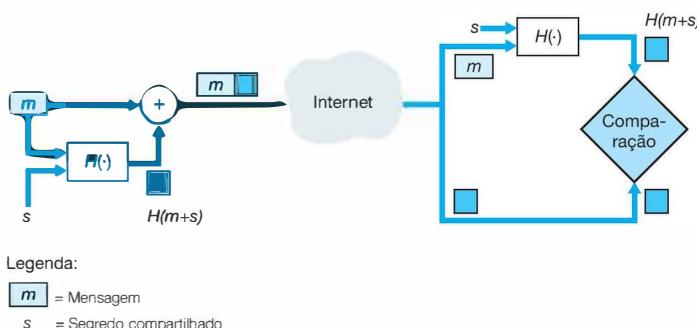
8.3.2 Código de autenticação da mensagem

Retornemos ao problema da integridade da mensagem. Agora que compreendemos as funções de *hash*, vamos tentar entender como podemos garantir a integridade da mensagem:

1. Alice cria a mensagem m e calcula o *hash* $H(m)$ (por exemplo, com SHA-1).
2. Alice então anexa $H(m)$ à mensagem m, criando uma mensagem estendida $(m, H(m))$, e a envia para Bob.
3. Bob recebe uma mensagem estendida (m, h) e calcula $H(m)$. Se $H(m) = h$, Bob conclui que está tudo certo.

Essa abordagem é, obviamente, errônea. Trudy pode criar uma mensagem m' falsa, passar-se por Alice, calcular $H(m')$ e enviar $(m', H(m'))$ a Bob. Quando Bob receber a mensagem, tudo se encaixa na etapa 3, então ele não suspeita de nada.

Para realizar a integridade da mensagem, além de usar as funções de *hash* criptográficas, Alice e Bob precisarão de um segredo compartilhado *s*, que não é nada mais do que uma cadeia de bits denominada **chave de autenticação**. Utilizando esse segredo compartilhado, a integridade da mensagem pode ser realizada da seguinte maneira:

1. Alice cria a mensagem *m*, concatena *s* com *m* para criar $m + s$, e calcula o *hash* $H(m+s)$ (por exemplo, com SHA-1). $H(m+s)$ é denominado o **código de autenticação da mensagem (MAC).**
2. Alice então anexa o MAC à mensagem *m*, criando uma mensagem estendida $(m, H(m+s))$, e a envia para Bob.
3. Bob recebe uma mensagem estendida (m, h) e, conhecendo *s*, calcula o MAC $H(m+s)$. Se $H(m+s) = h$, Bob conclui que está tudo certo.

Um resumo desse processo é ilustrado na Figura 8.9. É importante observar que o MAC (abreviação de *message authentication code* [código de autenticação da mensagem]), neste caso, não é o mesmo MAC utilizado nos protocolos da camada de enlace (abreviação de *medium access control* [controle de acesso ao meio])!

Um bom recurso do MAC é o fato de ele não exigir um algoritmo de criptografia. De fato, em muitas aplicações, incluindo o algoritmo de roteamento de estado do enlace, descrito antes, as entidades de comunicação somente estão preocupadas com a integridade da mensagem e não com o seu sigilo. Utilizando um MAC, as entidades podem autenticar as mensagens que enviam uma à outra sem ter de integrar algoritmos de criptografia complexos ao processo de integridade.

Como você pode esperar, muitos padrões diferentes para os MACs foram propostos ao longo dos anos. Hoje, o mais popular é o **HMAC**, que pode ser usado com o MD5 ou SHA-1. O HMAC, na verdade, passa os dados e a chave de autenticação duas vezes pela função de *hash* [Kaufman, 1995; RFC 2104].

Ainda resta um assunto importante. Como distribuímos a chave de autenticação compartilhada às entidades de comunicação? No algoritmo de roteamento de estado do enlace, por exemplo, precisaríamos, de alguma forma, distribuir a chave de autenticação secreta a cada um dos roteadores no sistema independente. (Observe que os roteadores podem usar a mesma chave de autenticação.) Um administrador de rede conseguiria, de fato, esse feito visitando fisicamente cada um dos roteadores. Ou, se o administrador de rede for preguiçoso, e se cada roteador possuir sua própria chave pública, ele poderia distribuir a chave de autenticação a qualquer um dos roteadores criptografando-a com a chave pública e, então, enviar a chave cifrada por meio da rede ao roteador.

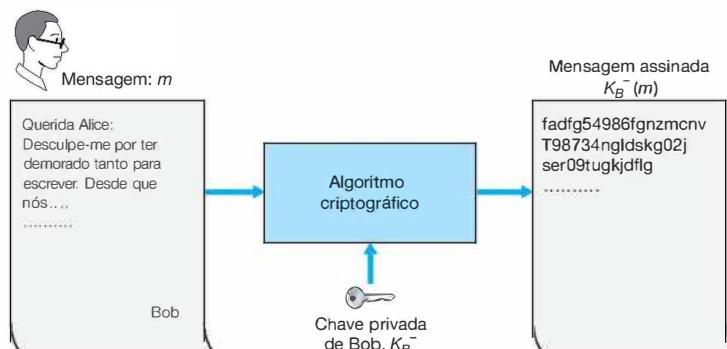
FIGURA 8.9 CÓDIGO DE AUTENTICAÇÃO DE MENSAGEM (MAC)

8.3.3 Assinaturas digitais

Pense no número de vezes em que você assinou seu nome em um pedaço de papel durante a última semana. Você assina cheques, comprovantes de operação de cartões de crédito, documentos legais e cartas. Sua assinatura atesta o fato de que você (e não outra pessoa) conhece o conteúdo do documento e/ou concorda com ele. No mundo digital, com frequência deseja-se indicar o dono ou o criador de um documento ou deixar claro que alguém

concorda com o conteúdo de um documento. A **assinatura digital** é uma técnica criptográfica que cumpre essas finalidades no mundo digital.

Exatamente como acontece com as assinaturas por escrito, a assinatura digital deve ser verificável e não falsificável. Isto é, deve ser possível provar que um documento assinado por um indivíduo foi na verdade assinado por ele (a assinatura tem de ser verificável) e que somente aquele indivíduo poderia ter assinado o documento (a assinatura não pode ser falsificada).

Vamos considerar agora como podemos criar um método de assinatura digital. Observe que, quando Bob assina uma mensagem, deve colocar algo nela que seja único para ele. Bob poderia adicionar um MAC à assinatura, sendo o MAC criado ao adicionar sua chave (única para ele) à mensagem e, depois, formar o *hash*. Mas, para Alice verificar a assinatura, ela deve também ter uma cópia da chave, que não seria única para Bob. Portanto, os MACs não se incluem nesse processo.

Lembre-se de que, com a criptografia de chave pública, Bob possui tanto uma chave pública como uma privada, as quais são únicas para ele. Dessa maneira, a criptografia de chave pública é uma excelente candidata para prover assinaturas digitais. Vamos verificar como isso é feito.

Suponha que Bob queira assinar digitalmente um documento, m. Imagine que o documento seja um arquivo ou uma mensagem que ele vai assinar e enviar. Como mostra a Figura 8.10, para assinar esse documento Bob apenas usa sua chave criptográfica privada K_B^- para processar $K_B^-(m)$. A princípio, pode parecer estranho que Bob esteja usando sua chave privada (que, como vimos na Seção 8.2, foi usada para decriptar uma mensagem que tinha sido criptografada com sua chave pública) para assinar um documento. Mas lembre-se de que criptografia e decriptação nada mais são do que uma operação matemática (exponenciação à potência e ou d no RSA; veja a Seção 8.2) e que a intenção de Bob não é embaralhar ou disfarçar o conteúdo do documento, mas assiná-lo de maneira que este seja verificável, não falsificável e incontestável. Bob tem o documento, m, e sua assinatura digital do documento é $K_B^-(m)$.

A assinatura digital $K_B^-(m)$ atende às exigências de ser verificável e não falsificável? Suponha que Alice tenha m e $K_B^-(m)$. Ela quer provar na Justiça (em ação litigiosa) que Bob de fato assinou o documento e que ele era a única pessoa que poderia tê-lo assinado. Alice pega a chave pública de Bob, $K_B^+(m)$, e lhe aplica a assinatura digital $K_B^-(m)$ associada ao documento m. Isto é, ela processa $K_B^+(K_B^-(m))$, e, *voilà*, com dramática encenação, produz m, que é uma reprodução exata do documento original! Ela então argumenta que somente Bob poderia ter assinado o documento pelas seguintes razões:

- Quem quer que tenha assinado o documento deve ter usado a chave criptográfica privada, K_B^-, para processar a assinatura $K_B^-(m)$, de modo que $K_B^+(K_B^-(m)) = m$.
- A única pessoa que poderia conhecer a chave privada, K_B^-, é Bob. Lembre-se de que dissemos em nossa discussão do RSA, na Seção 8.2, que conhecer a chave pública, K_B^+, não serve para descobrir a chave privada, K_B^-. Portanto, a única pessoa que poderia conhecer K_B^- é aquela que gerou o par de chaves (K_B^+, K_B^-) em

FIGURA 8.10 CRIANDO UMA ASSINATURA DIGITAL PARA UM DOCUMENTO

primeiro lugar, Bob. (Note que, para isso, admitimos que Bob não passou K_B^- a ninguém e que ninguém roubou K_B^- de Bob.)

Também é importante notar que, se o documento original, *m*, for modificado para alguma forma alternativa, *m*′, a assinatura que Bob criou para *m* não será válida para *m*′, já que $K_B^+(K_B^-(m))$ não é igual a *m*′. Assim, observamos que as assinaturas digitais também oferecem integridade da mensagem, permitindo que o receptor verifique se ela foi alterada, bem como sua origem.

Uma preocupação com os dados de assinatura é que a criptografia e a decriptação prejudicam o desempenho do computador. Sabendo do trabalho extra de criptografia e decriptação, os dados de assinatura por meio de criptografia/decriptação completa podem ser excessivos. Uma técnica mais eficiente é introduzir as funções de *hash* na assinatura digital. Na Seção 8.3.2, um algoritmo de *hash* pega uma mensagem, *m*, de comprimento qualquer e calcula uma "impressão digital" de comprimento fixo da mensagem, representada por *H*(*m*). Usando uma função de *hash*, Bob assina o *hash* da mensagem em vez de assinar a própria mensagem, ou seja, Bob calcula $K_B^-(H(m))$. Como *H*(*m*) é em geral muito menor do que a mensagem original *m*, o esforço computacional necessário para criar a assinatura digital é reduzido consideravelmente.

Em relação a Bob enviar uma mensagem para Alice, a Figura 8.11 apresenta um resumo do procedimento operacional de criar uma assinatura digital. Bob coloca sua longa mensagem original em uma função de *hash*. Ele, então, assina digitalmente o *hash* resultante com sua chave privada. A mensagem original (em texto claro) junto com o resumo de mensagem assinada digitalmente (de agora em diante denominada assinatura digital) é então enviada para Alice. A Figura 8.12 apresenta um resumo do processo operacional da assinatura. Alice aplica a função de *hash* à mensagem para obter um resultado de *hash*. Ela também aplica a função de *hash* à mensagem em texto claro para obter um segundo resultado de *hash*. Se os dois combinarem, então Alice pode ter certeza sobre a integridade e o autor da mensagem.

Antes de seguirmos em frente, vamos fazer uma breve comparação entre as assinaturas digitais e os MACs, visto que eles são paralelos, mas também têm diferenças sutis e importantes. As assinaturas digitais e os MACs iniciam com uma mensagem (ou um documento). Para criar um MAC por meio de mensagem, inserimos uma chave de autenticação à mensagem e, depois, pegamos o *hash* do resultado. Observe que nem a chave pública nem a criptografia de

FIGURA 8.11 ENVIANDO UMA MENSAGEM ASSINADA DIGITALMENTE

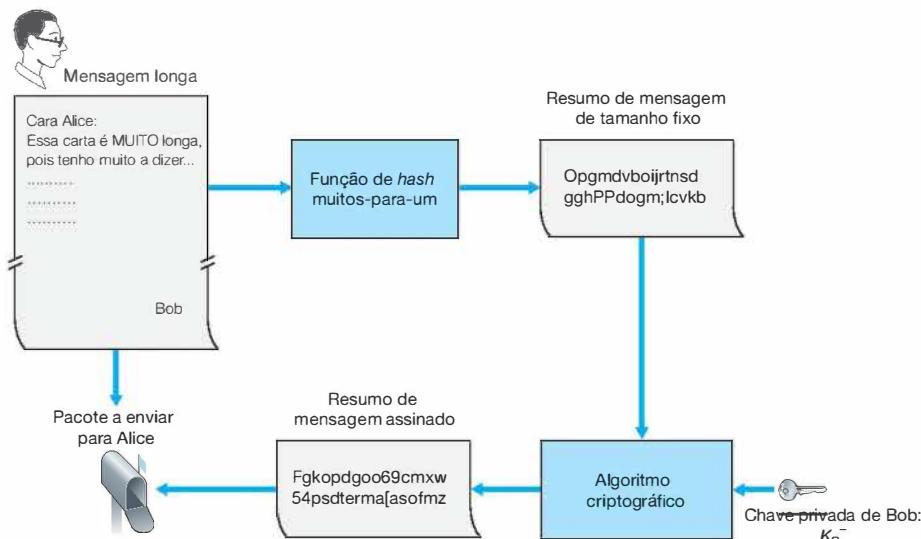
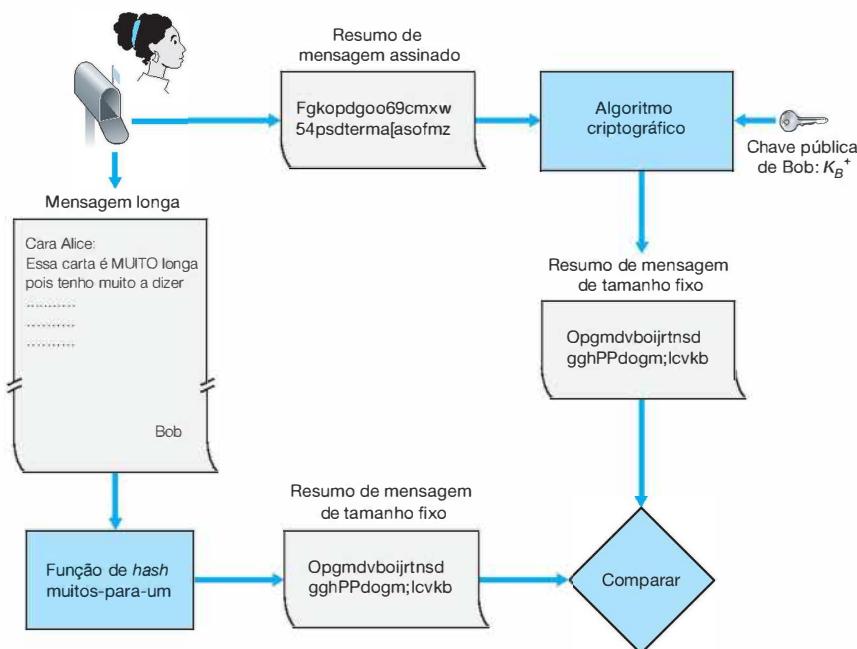



FIGURA 8.12 VERIFICANDO UMA MENSAGEM ASSINADA

chave simétrica estão envolvidas na criação do MAC. Para criar uma assinatura digital, primeiro pegamos o *hash* da mensagem e, depois, ciframos a mensagem com nossa chave privada (usando a criptografia de chave pública). Assim, uma assinatura digital é uma técnica "mais pesada", pois exige uma Infraestrutura de Chave Pública (PKI) subjacente com autoridades de certificação, conforme descrito abaixo. Veremos na Seção 8.4 que o PGP — um sistema seguro de e-mail e popular — utiliza assinaturas digitais para a integridade da mensagem. Já vimos que o OSPF usa MACs para a integridade da mensagem. Veremos nas seções 8.5 e 8.6 que os MACs são também usados por conhecidos protocolos de segurança da camada de transporte e da camada de rede.

Certificação de chaves públicas

Uma aplicação importante de assinaturas digitais é a **certificação de chaves públicas**, ou seja, certificar que uma chave pública pertence a uma entidade específica. A certificação de chaves públicas é usada em muitos protocolos de rede seguros, incluindo o IPsec e o SSL.

Para compreender mais sobre o problema, consideremos uma versão de comércio eletrônico para o clássico "trote da pizza". Suponha que Alice trabalhe no ramo de pizzas para viagem e que aceite pedidos pela Internet. Bob, que adora pizza, envia a Alice uma mensagem em texto aberto que contém o endereço de sua casa e o tipo de pizza que quer. Nessa mensagem, ele inclui também uma assinatura digital (isto é, um *hash* assinado extraído da mensagem original em texto aberto) para provar a Alice que ele é a origem verdadeira da mensagem. Para verificar a assinatura, Alice obtém a chave pública de Bob (talvez de um servidor de chaves públicas ou de uma mensagem de e-mail) e verifica a assinatura digital. Dessa maneira, ela se certifica de que foi Bob, e não algum adolescente brincalhão, quem fez o pedido.

Tudo parece caminhar bem até que entra em cena a esperta Trudy. Como mostrado na Figura 8.13, Trudy decide fazer uma travessura. Ela envia uma mensagem a Alice na qual diz que é Bob, fornece o endereço de Bob e

pede uma pizza. Nessa mensagem ela também inclui sua (de Trudy) chave pública, embora Alice suponha, claro, que seja a de Bob. Trudy também anexa uma assinatura digital, a qual foi criada com sua própria (de Trudy) chave privada. Após receber a mensagem, Alice aplica a chave pública de Trudy (pensando que é a de Bob) à assinatura digital e concluirá que a mensagem em texto aberto foi, na verdade, criada por Bob. Este ficará muito surpreso quando o entregador aparecer em sua casa com uma pizza de calabresa e enchovas!

Por esse exemplo, vemos que, para que a criptografia de chaves públicas seja útil, você precisa ser capaz de verificar se tem a chave pública verdadeira da entidade (pessoa, roteador, navegador e outras) com a qual quer se comunicar. Por exemplo, quando Alice estiver se comunicando com Bob usando criptografia de chaves públicas, ela precisará saber, com certeza, que a chave pública que supostamente é de Bob é, de fato, dele.

A vinculação de uma chave pública a uma entidade particular é feita, em geral, por uma **Autoridade Certificadora** (*Certification Authority* — CA), cuja tarefa é validar identidades e emitir certificados. Uma CA tem as seguintes funções:

1. Uma CA verifica se uma entidade (pessoa, roteador e assim por diante) é quem diz ser. Não há procedimentos obrigatórios para o modo como deve ser feita a certificação. Quando tratamos com uma CA, devemos confiar que ela tenha realizado uma verificação de identidade adequadamente rigorosa. Por exemplo, se Trudy conseguisse entrar na autoridade certificadora Fly-by-Night, e apenas declarasse "Eu sou Alice" e recebesse certificados associados à identidade de Alice, então não se deveria dar muita credibilidade a chaves públicas certificadas pela autoridade certificadora Fly-by-Night. Por outro lado, seria mais sensato (ou não!) estar mais inclinado a confiar em uma CA que faz parte de um programa federal ou estadual. O grau de confiança que se tem na identidade associada a uma chave pública equivale apenas ao grau de confiança depositada na CA e em suas técnicas de verificação de identidades. Que teia emaranhada de confiança estamos tecendo!
2. Tão logo verifique a identidade da entidade, a CA cria um **certificado** que vincula a chave pública da entidade à identidade verificada. O certificado contém a chave pública e a informação exclusiva que identifica mundialmente o proprietário da chave pública (por exemplo, o nome de alguém ou um endereço IP). O certificado é assinado digitalmente pela CA. Essas etapas são mostradas na Figura 8.14.

FIGURA 8.13 TRUDY SE PASSA POR BOB USANDO CRIPTOGRAFIA DE CHAVES PÚBLICAS

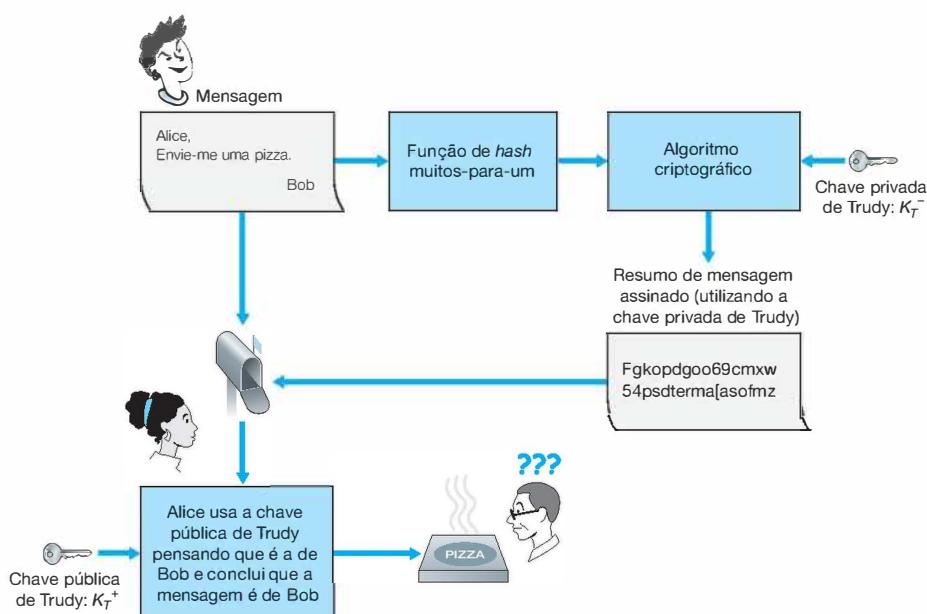
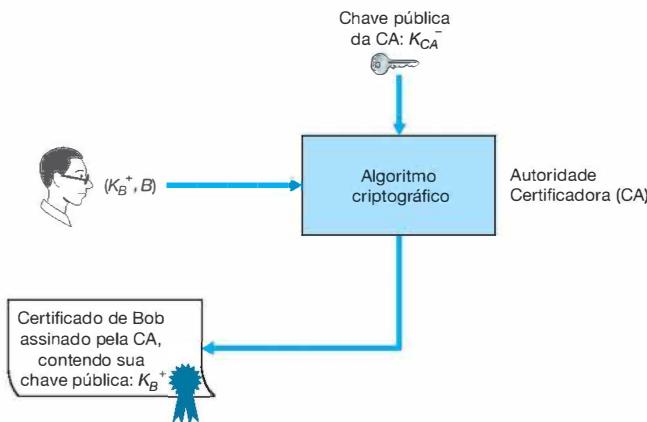



FIGURA 8.14 BOB OBTÉM SUA CHAVE PÚBLICA CERTIFICADA PELA CA

Vejamos, agora, como certificados podem ser usados para combater os espertinhos das pizzas, como Trudy e outros indesejáveis. Quando Bob faz seu pedido, ele também envia seu certificado assinado por uma CA. Alice usa a chave pública da CA para verificar a validade do certificado de Bob e extrair a chave pública dele.

Tanto a International Telecommunication Union (ITU) como a IETF desenvolveram padrões para autoridades certificadoras. Na recomendação ITU X.509 [ITU, 2005a], encontramos a especificação de um serviço de autenticação, bem como uma sintaxe específica para certificados. O RFC 1422 descreve um gerenciamento de chaves baseado em CA para utilização com o e-mail seguro pela Internet. Essa recomendação é compatível com X.509, mas vai além, pois estabelece procedimentos e convenções para uma arquitetura de gerenciamento de chaves. A Tabela 8.4 apresenta alguns campos importantes de um certificado.

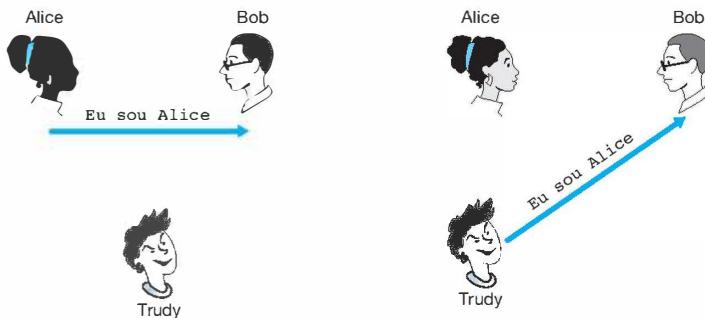
TABELA 8.4 CAMPOS SELECIONADOS DE UMA CHAVE PÚBLICA X.509 E RFC 1422

Nome do campo	Descrição
Versão	Número da versão da especificação X.509
Número de série	Identificador exclusivo emitido pela CA para um certificado
Assinatura	Especifica o algoritmo usado pela CA para assinar esse certificado
Nome do emissor	Identidade da CA que emitiu o certificado, em formato de nome distinto (DN) [RFC 2253]
Período de validade	Início e fim do período de validade do certificado
Nome do sujeito	Identidade da entidade cuja chave pública está associada a esse certificado, em formato DN
Chave pública do sujeito	A chave pública do sujeito, bem como uma indicação do algoritmo de chave pública (e parâmetros do algoritmo) a ser usado com essa chave

8.4 AUTENTICAÇÃO DO PONTO FINAL

A **autenticação do ponto final** é o processo de provar a identidade de uma entidade a outra entidade por uma rede de computadores, por exemplo, um usuário provando sua identidade a um servidor de correio eletrônico. Como seres humanos, autenticamos uns aos outros de diversas maneiras: reconhecemos o rosto do outro ao nos encontrarmos, reconhecemos a voz no telefone, somos autenticados por um oficial da Alfândega que compara a nossa foto com a do passaporte.

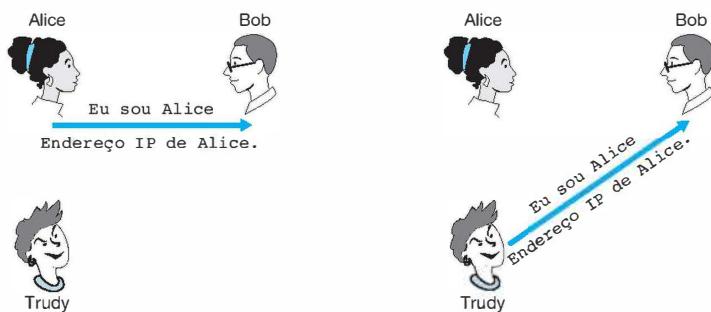
Nesta seção, vamos considerar como uma entidade pode autenticar outra quando os dois estão se comunicado por uma rede. Vamos nos atentar aqui para a autenticação de uma entidade "ao vivo", no momento em que a comunicação está de fato ocorrendo. Um exemplo concreto é um usuário autenticando-se em um servidor de correio eletrônico. Este é um problema sutilmente diferente de provar que uma mensagem recebida em algum ponto no passado veio mesmo do remetente afirmado, conforme estudamos na Seção 8.3.

Ao realizar autenticação pela rede, as entidades comunicantes não podem contar com informações biométricas, como aparência visual ou timbre de voz. Na verdade, veremos em nossos estudos de caso que, geralmente, são os elementos da rede como roteadores e processos cliente/servidor que precisam se autenticar. Aqui, a autenticação precisa ser feita unicamente com base nas mensagens e dados trocados como parte de um **protocolo de autenticação**. Em geral, um protocolo de autenticação seria executado *antes* que as duas entidades comunicantes executassem algum outro protocolo (por exemplo, um protocolo de transferência confiável de dados, um de troca de informações de roteamento ou um de correio eletrônico). O protocolo de autenticação primeiro estabelece as identidades das partes para a satisfação mútua; somente depois da autenticação é que as entidades "põem as mãos" no trabalho real.

Assim como no caso do nosso desenvolvimento de um protocolo de transferência confiável de dados (rdt) no Capítulo 3, será esclarecedor desenvolvermos aqui diversas versões de um protocolo de autenticação, que chamaremos de **ap** (*authentication protocol* — protocolo de autenticação), explicando cada versão enquanto prosseguimos. (Se você gostar da evolução passo a passo de um projeto, também poderá gostar de Bryant [1988], que relata uma narrativa fictícia entre projetistas de um sistema aberto de autenticação de rede e sua descoberta das muitas questões sutis que estão envolvidas.)

Vamos imaginar que Alice precise se autenticar com Bob.

8.4.1 Protocolo de autenticação *ap1.0*

Talvez o protocolo de autenticação mais simples que possamos imaginar seja aquele onde Alice apenas envia uma mensagem a Bob dizendo ser Alice. Esse protocolo aparece na Figura 8.15. A falha aqui é óbvia — não há como Bob realmente saber que a pessoa enviando a mensagem "Eu sou Alice" é mesmo Alice. Por exemplo, Trudy (a intrusa) poderia enviar tal mensagem.

FIGURA 8.15 PROTOCOLO *ap1.0* E UM CENÁRIO DE FALHA

8.4.2 Protocolo de autenticação *ap2.0*

Se Alice possui um endereço de rede conhecido (por exemplo, um endereço IP) do qual ela sempre se comunica, Bob poderia tentar autenticar Alice verificando se o endereço de origem no datagrama IP que transporta a mensagem de autenticação combina com o endereço conhecido de Alice. Nesse caso, Alice seria autenticada. Isso poderia impedir que um intruso muito ingênuo em redes se passe por Alice, mas não o aluno determinado, que está estudando este livro, ou muitos outros!

Pelo estudo das camadas de rede e enlace de dados, sabemos que não é difícil (por exemplo, se alguém tivesse acesso ao código do sistema operacional e pudesse criar seu próprio núcleo do sistema operacional, como acontece com o Linux e vários outros sistemas operacionais disponíveis livremente) criar um datagrama IP, colocar qualquer endereço de origem IP que se desejar (por exemplo, o endereço IP conhecido de Alice) no datagrama IP e enviar o datagrama pelo protocolo da camada de enlace para o roteador do primeiro salto. Daí em diante, o datagrama com endereço de origem incorreto seria zelosamente repassado para Bob. Essa técnica, mostrada na Figura 8.16, é uma forma de falsificação de IP. Tal falsificação pode ser evitada se o roteador do primeiro salto de Trudy for configurado para encaminhar apenas datagramas contendo o endereço IP de origem de Trudy [RFC 2827]. Porém, essa capacidade não é implementada ou imposta de modo universal. Assim, Bob poderia ser enganado a achar que o gerente de rede de Trudy (que poderia ser a própria Trudy) configurou o roteador do primeiro salto de Trudy para repassar somente datagramas corretamente endereçados.

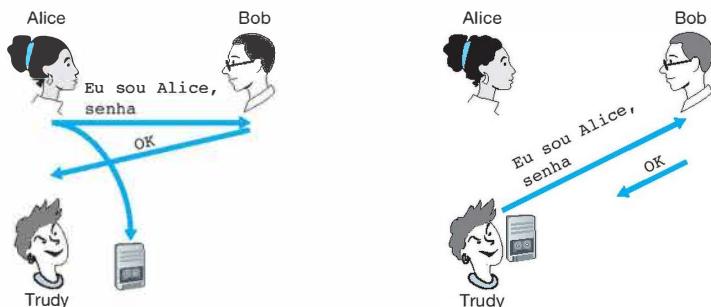



FIGURA 8.16 PROTOCOLO *ap2.0* E UM CENÁRIO DE FALHA

8.4.3 Protocolo de autenticação *ap3.0*

Uma técnica clássica de autenticação é usar uma senha secreta. A senha é um segredo compartilhado entre o autenticador e a pessoa sendo autenticada. Gmail, Facebook, telnet, FTP e muitos outros serviços usam a autenticação por senha. No protocolo *ap3.0*, Alice envia sua senha secreta a Bob, como mostra a Figura 8.17.

FIGURA 8.17 PROTOCOLO *ap3.0* E UM CENÁRIO DE FALHA

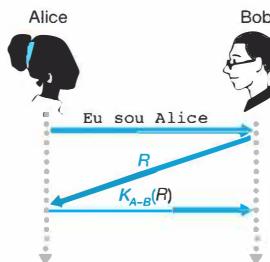
Como as senhas são muito utilizadas, poderíamos supor que o protocolo *ap3.0* é bastante seguro. Nesse caso, estaríamos errados! A falha de segurança aqui é clara. Se Trudy interceptar a comunicação de Alice, então ela poderá descobrir a senha de Alice. Se você acha que isso é improvável, considere o fato de que, quando você usa Telnet com outra máquina e se autentica, a senha de autenticação é enviada abertamente para o servidor Telnet. Alguém conectado ao cliente Telnet ou à LAN do servidor possivelmente poderia investigar (ler e armazenar) todos os pacotes transmitidos na LAN e, assim, roubar a senha de autenticação. De fato, essa é uma técnica bem conhecida para roubar senhas (veja, por exemplo, Jimenez [1997]). Essa ameaça, obviamente, é bastante real, de modo que *ap3.0* certamente não funcionará.

8.4.4 Protocolo de autenticação *ap3.1*

Nossa próxima ideia para consertar o *ap3.0*, naturalmente, é criptografar a senha. Criptografando a senha, podemos impedir que Trudy descubra a senha de Alice. Se considerarmos que Alice e Bob compartilham uma chave secreta simétrica, K_{A-B}, então Alice pode criptografar a senha e enviar sua mensagem de identificação, "`Eu sou Alice`", e sua senha criptografada para Bob. Então, Bob decodifica a senha e, supondo que a senha esteja correta, autentica Alice. Bob está confiante na autenticação de Alice, pois Alice não apenas conhece a senha, mas também sabe o valor da chave secreta compartilhada necessária para criptografar a senha. Vamos chamar esse protocolo de *ap3.1*.

Embora seja verdade que *ap3.1* impede que Trudy descubra a senha de Alice, o uso da criptografia aqui não resolve o problema da autenticação. Bob está sujeito a um **ataque de reprodução**: Trudy só precisa investigar a comunicação de Alice, gravar a versão criptografada da senha e reproduzir a versão criptografada da senha para Bob, fingindo ser Alice. O uso de uma senha criptografada em *ap3.1* não torna a situação muito diferente daquela do protocolo *ap3.0*, na Figura 8.17.

8.4.5 Protocolo de autenticação *ap4.0*

O cenário de falha na Figura 8.17 resultou do fato de Bob não conseguir distinguir entre a mensagem original enviada por Alice e a reprodução dessa mensagem. Ou seja, Bob não conseguiu saber se Alice estava ao vivo (isto é, realmente estava no outro ponto da conexão) ou se a mensagem que ele recebeu foi uma reprodução de uma autenticação anterior. O leitor muito (*muito*) atento se lembrará de que o protocolo de apresentação de três vias precisou resolver o mesmo problema — o lado do servidor de uma conexão TCP não queria aceitar uma conexão se o segmento SYN recebido fosse uma cópia antiga (retransmissão) de um segmento SYN de uma conexão anterior. Como o lado do servidor TCP resolveu o problema de determinar se o cliente estava mesmo ao vivo? Ele escolheu um número de sequência inicial que não havia sido usado por um bom tempo, enviou esse número ao cliente, e então aguardou que o cliente respondesse com um segmento ACK contendo esse número. Podemos adotar a mesma ideia para fins de autenticação.

Um ***nonce*** é um número que o protocolo usará somente uma vez por toda a vida. Ou seja, uma vez que o protocolo utilizar um *nonce*, esse número nunca mais será utilizado. Nosso protocolo *ap4.0* usa um *nonce* da seguinte forma:

1. Alice envia a mensagem "`Eu sou Alice`" para Bob.
2. Bob escolhe um *nonce*, *R*, e o envia a Alice.
3. Alice criptografa o *nonce* usando a chave secreta simétrica de Alice e Bob, K_{A-B}, e envia o *nonce* criptografado, $K_{A-B}(R)$, de volta a Bob. Assim como no protocolo *ap3.1*, o fato de Alice conhecer K_{A-B} e o utilizar para criptografar o valor permite a Bob saber que a mensagem recebida foi gerada por Alice. O *nonce* é usado para garantir que Alice está ao vivo.
4. Bob decodifica a mensagem recebida. Se o *nonce* decodificado for igual ao que ele enviou a Alice, então Alice está autenticada.

O protocolo *ap4.0* é ilustrado na Figura 8.18. Usando o valor exclusivo de R e depois verificando o valor retornado, $K_{A-B}(R)$, Bob pode estar certo de que Alice tanto é quem ela diz ser (pois conhece o valor da chave secreta necessária para criptografar R) quanto está ao vivo (pois criptografou o *nonce*, R, que Bob acabou de criar).

O uso de um *nonce* e das formas de criptografia por chave simétrica forma a base do *ap4.0*. Uma pergunta natural é se podemos usar um *nonce* e a criptografia de chave pública (em vez da de chave simétrica) para resolver o problema de autenticação. Essa questão é explorada nos problemas ao final deste capítulo.

FIGURA 8.18 PROTOCOLO *ap4.0* E UM CENÁRIO DE FALHA

8.5 PROTEGENDO O E-MAIL

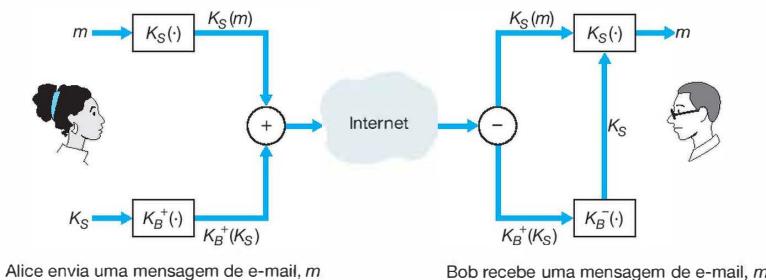
Nas seções anteriores, analisamos as questões fundamentais em segurança de redes, incluindo a criptografia de chave simétrica e de chave pública, autenticação do ponto final, distribuição de chave, integridade da mensagem e assinaturas digitais. Agora vamos analisar como essas ferramentas estão sendo usadas para oferecer segurança na Internet.

Curiosamente, é possível oferecer serviços de segurança em cada uma das quatro principais camadas da pilha de protocolos da Internet. Quando a segurança é fornecida para um protocolo específico da camada de aplicação, a aplicação que o emprega utilizará um ou mais serviços de segurança, como sigilo, autenticação ou integridade. Quando é oferecida para um protocolo da camada de transporte, todas as aplicações que usam esse protocolo aproveitam os seus serviços de segurança. Quando a segurança é fornecida na camada de rede em base hospedeiro para hospedeiro, todos os segmentos da camada de transporte (e, portanto, todos os dados da camada de aplicação) aproveitam os serviços de segurança dessa camada. Quando a segurança é oferecida em um enlace, os dados em todos os quadros que percorrem o enlace recebem os serviços de segurança do enlace.

Nas seções 8.5 a 8.8, verificamos como as ferramentas de segurança estão sendo usadas nas camadas de enlace, rede, transporte e aplicação. Obedecendo à estrutura geral deste livro, começamos no topo da pilha de protocolo e discutimos segurança na camada de aplicação. Nossa técnica é usar uma aplicação específica, e-mail, como um estudo de caso para a segurança da camada de aplicação. Então, descemos a pilha de protocolo. Examinaremos o protocolo SSL (que provê segurança na camada de transporte), o IPsec (que provê segurança na camada de rede) e a segurança do protocolo LAN IEEE 802.11 sem fio.

Você deve estar se perguntando por que a funcionalidade da segurança está sendo fornecida em mais de uma camada na Internet. Já não bastaria prover essa funcionalidade na camada de rede? Há duas respostas para a pergunta. Primeiro, embora a segurança na camada de rede possa oferecer "cobertura total" cifrando todos os dados nos datagramas (ou seja, todos os segmentos da camada de transporte) e autenticando todos os endereços IP destinatários, ela não pode prover segurança no nível do usuário. Por exemplo, um site de comércio não pode confiar na segurança da camada IP para autenticar um cliente que vai comprar mercadorias. Assim, existe a necessidade de uma funcionalidade da segurança em camadas superiores bem como cobertura total em canais inferiores. Segundo, em geral é mais fácil implementar serviços da Internet, incluindo os de segurança nas camadas superiores da pilha de protocolo. Enquanto aguardamos a ampla implementação da segurança na camada de

rede, o que ainda levará muitos anos, muitos criadores de aplicação "já fazem isso" e introduzem a funcionalidade da segurança em suas aplicações favoritas. Um exemplo clássico é o Pretty Good Privacy (PGP), que oferece e-mail seguro (discutido mais adiante nesta seção). Necessitando de apenas um código de aplicação do cliente e do servidor, o PGP foi uma das primeiras tecnologias de segurança a ser amplamente utilizada na Internet.

8.5.1 E-mail seguro

Agora usamos os princípios de criptografia das seções 8.2 a 8.3 para criar um sistema de e-mail seguro. Criamos esse projeto de alto nível de maneira incremental, introduzindo, a cada etapa, novos serviços de segurança. Em nosso projeto de um sistema de e-mail seguro, vamos manter em mente o exemplo picante apresentado na Seção 8.1 — o caso de amor entre Alice e Bob. Imagine que Alice quer enviar uma mensagem de e-mail para Bob e Trudy quer bisbilhotar.

Antes de avançar e projetar um sistema de e-mail seguro para Alice e Bob, devemos considerar quais características de segurança seriam as mais desejáveis para eles. A primeira, e mais importante, é a confidencialidade. Como foi discutido na Seção 8.1, nem Alice nem Bob querem que Trudy leia a mensagem de e-mail de Alice. A segunda característica que Alice e Bob provavelmente gostariam de ver no sistema de e-mail seguro é a autenticação do remetente. Em particular, quando Bob receber a seguinte mensagem: "`Eu não o amo mais. Nunca mais quero vê-lo. Da anteriormente sua, Alice`", ele naturalmente gostaria de ter certeza de que a mensagem veio de Alice, e não de Trudy. Outra característica de segurança de que os dois amantes gostariam de dispor é a *integridade de mensagem*, isto é, a certeza de que a mensagem que Alice enviar não será modificada no trajeto até Bob. Por fim, o sistema de e-mail deve fornecer *autenticação do receptor*, isto é, Alice quer ter certeza de que de fato está enviando a mensagem para Bob, e não para outra pessoa (por exemplo, Trudy) que esteja se passando por ele.

Portanto, vamos começar abordando a preocupação mais premente, a confidencialidade. A maneira mais direta de conseguir confidencialidade é Alice criptografar a mensagem com tecnologia de chaves simétricas (como DES ou AES) e Bob decriptar a mensagem ao recebê-la. Como discutido na Seção 8.2, se a chave simétrica for longa o suficiente e se apenas Alice e Bob possuírem a chave, então será dificílimo que alguém (incluindo Trudy) leia a mensagem. Embora essa seja uma abordagem direta, ela apresenta a dificuldade fundamental que discutimos na Seção 8.2 — é difícil distribuir uma chave simétrica de modo que apenas Bob e Alice tenham cópias dela. Portanto, é natural que consideremos uma abordagem alternativa — a criptografia de chaves públicas (usando, por exemplo, RSA). Nessa abordagem, Bob disponibiliza publicamente sua chave pública (por exemplo, em um servidor de chaves públicas ou em sua página pessoal) e Alice criptografa sua mensagem com a chave pública de Bob, e envia a mensagem criptografada para o endereço de e-mail de Bob. Quando recebe a mensagem, ele simplesmente a decodifica com sua chave privada. Supondo que Alice tenha certeza de que aquela chave pública é a de Bob, essa técnica é um meio excelente de fornecer a confidencialidade desejada. Um problema, contudo, é que a criptografia de chaves públicas é relativamente ineficiente, sobretudo para mensagens longas.

Para superar o problema da eficiência, vamos fazer uso de uma chave de sessão (discutida na Seção 8.2.2). Em particular, Alice (1) escolhe uma chave simétrica, K_S, aleatoriamente, (2) criptografa sua mensagem m com a chave simétrica, *Ks,* (3) criptografa a chave simétrica com a chave pública de Bob, K_B^+, (4) concatena a mensagem criptografada e a chave simétrica criptografada de modo que formem um "pacote" e (5) envia o pacote ao endereço de e-mail de Bob. Os passos estão ilustrados na Figura 8.19. (Nessa figura e nas subsequentes, o sinal "+" dentro de um círculo representa formar a concatenação e o sinal "–" dentro de um círculo, desfazer a concatenação.) Quando Bob receber o pacote, ele (1) usará sua chave privada K_B^-, para obter a chave simétrica K_S, e (2) utilizará a chave simétrica K_S para decodificar a mensagem m.

Agora que projetamos um sistema de e-mail seguro que fornece confidencialidade, vamos desenvolver um outro sistema que forneça autenticação do remetente e também integridade de mensagem. Vamos supor, por enquanto, que Alice e Bob não estejam mais preocupados com confidencialidade (querem compartilhar seus sentimentos com todos!) e que só estejam preocupados com a autenticação do remetente e com a integridade da mensagem. Para realizar essa tarefa, usaremos assinaturas digitais e resumos de mensagem, como descrito na

FIGURA 8.19 ALICE USOU UMA CHAVE DE SESSÃO SIMÉTRICA, K_S, PARA ENVIAR UM E-MAIL SECRETO PARA BOB

Seção 8.3. Especificamente, Alice (1) aplica uma função de *hash* H (por exemplo, MD5) à sua mensagem *m*, para obter um resumo, (2) assina o resultado da função de *hash* com sua chave privada K_A^- para criar uma assinatura digital, (3) concatena a mensagem original (não criptografada) com a assinatura para criar um pacote e (4) envia o pacote ao endereço de e-mail de Bob. Quando Bob recebe o pacote, ele (1) aplica a chave pública de Alice, K_A^+, ao resumo de mensagem assinado e (2) compara o resultado dessa operação com o próprio *hash H* da mensagem. As etapas são ilustradas na Figura 8.20. Como discutimos na Seção 8.3, se os dois resultados forem iguais, Bob poderá ter razoável certeza de que a mensagem veio de Alice e não foi alterada.

Vamos considerar agora o projeto de um sistema de e-mail que forneça confidencialidade, autenticação de remetente *e* integridade de mensagem. Isso pode ser feito pela combinação dos procedimentos das figuras 8.19 e 8.20. Primeiro, Alice cria um pacote preliminar, exatamente como ilustra a Figura 8.20, constituído de sua mensagem original junto com um *hash* da mensagem assinado digitalmente. Em seguida, ela trata esse pacote preliminar como uma mensagem em si e a envia seguindo as etapas do remetente mostradas na Figura 8.19, criando um novo pacote, que é enviado a Bob. As etapas seguidas são mostradas na Figura 8.21. Quando Bob recebe o pacote, ele aplica primeiro seu lado da Figura 8.19 e depois seu lado da Figura 8.20. É preciso ficar claro que esse projeto atinge o objetivo de fornecer confidencialidade, autenticação de remetente e integridade de mensagem. Note que nesse esquema Alice utiliza criptografia de chaves públicas duas vezes: uma vez com sua chave privada e uma vez com a chave pública de Bob. De maneira semelhante, Bob também aplica a criptografia de chaves públicas duas vezes — uma vez com sua chave privada e uma vez com a chave pública de Alice.

O projeto de e-mail seguro ilustrado na Figura 8.21 provavelmente fornece segurança satisfatória para os usuários de e-mail na maioria das ocasiões. Mas ainda resta uma questão importante a ser abordada. O projeto da Figura 8.21 requer que Alice obtenha a chave pública de Bob e que este obtenha a chave pública de Alice. A distribuição dessas chaves é um problema nada trivial. Por exemplo, Trudy poderia se disfarçar de Bob e

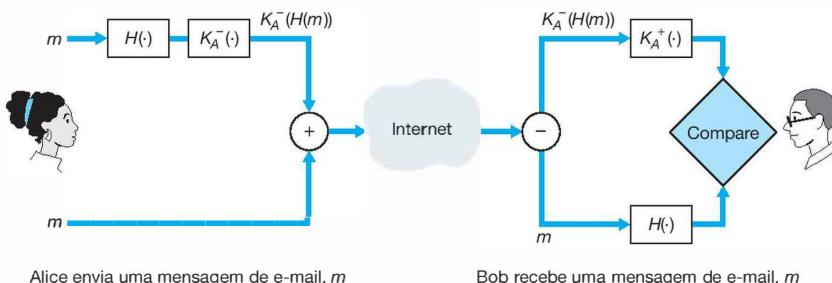
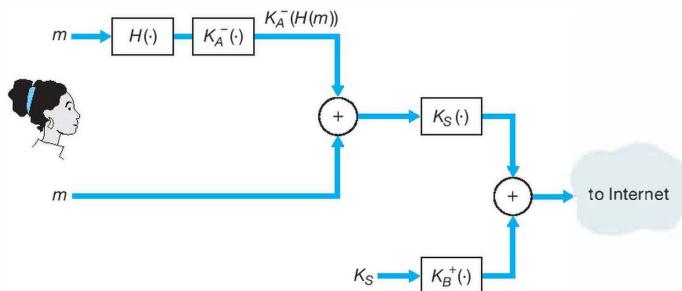



FIGURA 8.20 USANDO FUNÇÕES DE *HASH* E ASSINATURAS DIGITAIS PARA FORNECER AUTENTICAÇÃO DE REMETENTE E INTEGRIDADE DE MENSAGEM

FIGURA 8.21 ALICE USA CRIPTOGRAFIA DE CHAVES SIMÉTRICAS, CRIPTOGRAFIA DE CHAVES PÚBLICAS, UMA FUNÇÃO DE *HASH* E UMA ASSINATURA DIGITAL PARA FORNECER SIGILO, AUTENTICAÇÃO DE REMETENTE E INTEGRIDADE DE MENSAGEM

dar a Alice sua própria chave pública, dizendo que é a de Bob. Como aprendemos na Seção 8.3, uma tática popular para distribuir chaves públicas com segurança é *certificar* as chaves públicas usando uma autoridade certificadora (CA).

8.5.2 PGP

Projetado originalmente por Phil Zimmermann em 1991, o PGP (**Pretty Good Privacy** — privacidade razoável) é um esquema de criptografia para e-mail que se tornou um padrão de fato. O site do PGP é acessado mais de um milhão de vezes por mês por usuários de 166 países [PGPI, 2012]. Versões do PGP estão disponíveis em domínio público; por exemplo, você pode encontrar o software PGP para sua plataforma favorita, bem como grande quantidade de material de leitura interessante, na home page internacional do PGP [PGPI, 2012]. (Um ensaio de particular interesse, escrito pelo autor do PGP, é encontrado em Zimmermann [2012].) O projeto do PGP é, basicamente, idêntico ao projeto apresentado na Figura 8.21. Dependendo da versão, o software do PGP usa MD5 ou SHA para processar o resumo de mensagem; CAST, DES triplo ou IDEA para criptografar chaves simétricas, e RSA para criptografar chaves públicas.

Quando o PGP é instalado, o software cria um par de chaves públicas para o usuário. A chave pública pode então ser colocada no site do usuário ou em um servidor de chaves públicas. A chave privada é protegida pelo uso de uma senha. A senha tem de ser informada todas as vezes que o usuário acessar a chave privada. O PGP oferece ao usuário a opção de assinar digitalmente a mensagem, criptografar a mensagem ou, ainda, ambas as opções: assinar digitalmente e criptografar a mensagem. A Figura 8.22 mostra uma mensagem PGP assinada. Essa mensagem aparece após o cabeçalho MIME. Os dados codificados da mensagem correspondem a $K_A^-(H(m))$, isto é, ao resumo de mensagem assinado digitalmente. Como discutimos antes, para que Bob verifique a integridade da mensagem, ele precisa ter acesso à chave pública de Alice.

FIGURA 8.22 UMA MENSAGEM PGP ASSINADA

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
Bob:
Can I see you tonight?
Passionately yours, Alice
-----BEGIN PGP SIGNATURE-----
Version: PGP for Personal Privacy 5.0
Charset: noconv
yhHJRHhGJGhgg/12EpJ+lo8gE4vB3mqJhFEvZP9t6n7G6m5Gw2
-----END PGP SIGNATURE-----
```

HISTÓRIA

Phil Zimmermann e o PGP

Philip R. Zimmermann é o criador do Pretty Good Privacy (PGP). Por causa disso, foi alvo de uma investigação criminal que durou três anos, porque o governo norte-americano sustentava que as restrições sobre a exportação de software de criptografia tinham sido violadas quando o PGP se espalhou por todo o mundo após sua publicação como software de uso livre em 1991. Após ter sido liberado como software compartilhado, alguém o colocou na Internet e estrangeiros o baixaram. Nos Estados Unidos, os programas de criptografia são classificados como artefatos militares por lei federal e não podem ser exportados.

A despeito da falta de financiamento, da inexistência de representantes legais, da falta de uma empresa para lhe dar apoio e das intervenções governamentais, mesmo assim o PGP se tornou o software de criptografia para e-mail mais usado no mundo. O mais estranho é que o governo dos Estados Unidos pode ter contribuído inadvertidamente para a disseminação do PGP por causa do caso Zimmermann.

O governo norte-americano arquivou o caso no início de 1996. O anúncio foi festejado pelos ativistas da Internet. O caso Zimmermann tinha se transformado na história de um inocente que lutava por seus direitos contra os desmandos de um governo poderoso. A desistência do governo foi uma notícia bem-vinda, em parte por causa da campanha em favor da censura na Internet desencadeada pelo Congresso e em parte por causa dos esforços feitos pelo FBI para conseguir maior amplitude para a espionagem governamental.

Após o governo ter arquivado o caso, Zimmermann fundou a PGP Inc., que foi adquirida pela Network Associates em dezembro de 1997. Zimmermann é agora membro do conselho da Network Associates, bem como consultor independente para assuntos de criptografia.

A Figura 8.23 mostra uma mensagem PGP secreta. Essa mensagem também aparece após o cabeçalho MIME. É claro que a mensagem em texto aberto não está incluída na de e-mail secreta. Quando um remetente (como Alice) quer não apenas a confidencialidade, mas também a integridade, o PGP contém uma mensagem como a da Figura 8.23 dentro daquela da Figura 8.22.

O PGP também fornece um mecanismo para certificação de chaves públicas, mas esse mecanismo é bem diferente daquele da CA mais convencional. As chaves públicas do PGP são certificadas por uma *rede de confiança*. A própria Alice pode certificar qualquer par chave/usuário quando ela achar que esse par realmente está correto. Além disso, o PGP permite que Alice declare que ela confia em outro usuário para atestar a autenticidade de mais chaves. Alguns usuários do PGP assinam reciprocamente suas chaves montando grupos de assinatura de chaves. Os usuários se reúnem fisicamente, trocam chaves públicas e certificam suas chaves reciprocamente, assinando-as com suas chaves privadas.

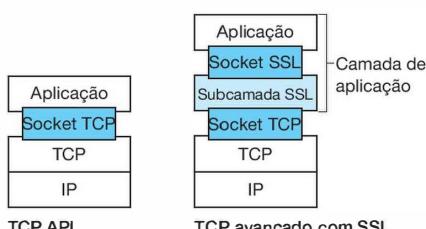
FIGURA 8.23 UMA MENSAGEM PGP SECRETA

```
-----BEGIN PGP MESSAGE-----
Version: PGP for Personal Privacy 5.0
u2R4d+/jKmn8Bc5+hgDsqAewsDfrGdszX68liKm5F6Gc4sDfcXyt
RfdS10juHgbcfDssWe7/K=lKhnMikLo0+1/BvcX4t==Ujk9PbcD4
Thdf2awQfgHbnmKlok8iy6gThlp
-----END PGP MESSAGE
```

8.6 PROTEGENDO CONEXÕES TCP: SSL

Na seção anterior, vimos como as técnicas criptográficas podem prover sigilo, integridade dos dados e autenticação do ponto final a aplicações específicas, ou seja, e-mail. Nesta, desceremos uma camada na pilha de

protocolo e examinaremos como a criptografia pode aprimorar o TCP com os serviços de segurança, incluindo sigilo, integridade dos dados e autenticação do ponto final. Essa versão aprimorada do TCP é denominada **Camada Segura de Sockets** (*Secure Sockets Layer — SSL*). Uma versão levemente modificada da versão 3 do SSL, denominada **Segurança na Camada de Transporte** (*Transport Layer Security* —TLS), foi padronizada pelo IETF [RFC 4346].

O SSL foi na origem projetado pela Netscape, mas as ideias básicas por trás da proteção do TCP antecedem o trabalho da Netscape (por exemplo, consulte Woo [1994]). Desde sua concepção, o SSL obteve uma ampla implementação. Ele é suportado por todos os navegadores Web e servidores Web, e usado basicamente por todos os sites populares (incluindo Amazon, eBay, Yahoo!, MSN etc.). Dezenas de bilhões de dólares são gastos com o SSL a cada ano. Na verdade, se você já comprou qualquer coisa pela Internet com seu cartão de crédito, a comunicação entre seu navegador e servidor para essa compra foi quase certamente por meio do SSL. (Você pode identificar que o SSL está sendo usado por seu navegador quando a URL se iniciar com https: em vez de http:.)

Para entender a necessidade do SSL, vamos examinar um cenário de comércio pela Internet típico. Bob está navegando na Web e acessa o site Alice Incorporated, que está vendendo perfume. O site Alice Incorporated exibe um formulário no qual Bob deve inserir o tipo de perfume e quantidade desejados, seu endereço e o número de seu cartão de crédito. Bob insere essas informações, clica em Enviar, e espera pelo recebimento (via correio postal comum) de seus perfumes; ele também espera pelo recebimento de uma cobrança pelo seu pedido na próxima fatura do cartão de crédito. Até o momento, tudo bem, mas se nenhuma medida de segurança for tomada, Bob poderia esperar por surpresas.

- Se nenhum sigilo (encriptação) for utilizado, um invasor poderia interceptar o pedido de Bob e receber suas informações sobre o cartão. O invasor poderia, então, fazer compras à custa de Bob.
- Se nenhuma integridade de dados for utilizada, um invasor poderia modificar o pedido de Bob, fazendo-o comprar dez vezes mais frascos de perfumes que o desejado.
- Finalmente, se nenhuma autenticação do servidor for utilizada, um servidor poderia exibir o famoso logotipo da Alice Incorporated, quando na verdade o site é mantido por Trudy, que está se passando por Alice Incorporated. Após receber o pedido de Bob, Trudy poderia ficar com o dinheiro dele e sumir. Ou Trudy poderia realizar um roubo de identidade obtendo o nome, endereço e número do cartão de crédito de Bob.

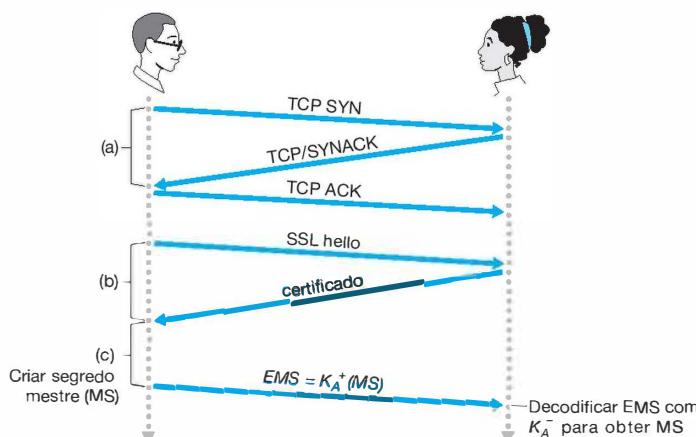
O SSL resolve essas questões aprimorando o TCP com sigilo, integridade dos dados, autenticação do servidor e autenticação do cliente.

Muitas vezes, o SSL é usado para oferecer segurança em transações que ocorrem pelo HTTP. Entretanto, como o SSL protege o TCP, ele pode ser empregado por qualquer aplicação que execute o TCP. O SSL provê uma Interface de Programação de Aplicação (API) com *sockets*, semelhante à API do TCP. Quando uma aplicação quer empregar o SSL, ela inclui classes/bibliotecas SSL. Como mostrado na Figura 8.24, embora o SSL resida tecnicamente na camada de aplicação, do ponto de vista do desenvolvedor, ele é um protocolo de transporte que provê serviços do TCP aprimorados com serviços de segurança.

FIGURA 8.24 EMBORA O SSL RESIDA TECNICAMENTE NA CAMADA DE APLICAÇÃO, DO PONTO DE VISTA DO DESENVOLVEDOR, ELE É UM PROTOCOLO DA CAMADA TRANSPORTE

8.6.1 Uma visão abrangente

Começamos descrevendo uma versão simplificada do SSL, que nos permitirá obter uma visão abrangente de *por que* e *como* funciona o SSL. Vamos nos referir a essa versão simplificada do SSL como "quase-SSL". Após descrever o quase-SSL, na próxima subseção, descreveremos o SSL de verdade, preenchendo as lacunas. O quase-SSL (e o SSL) possui três fases: *apresentação (handshake), derivação de chave* e *transferência de dados.* Agora descreveremos essas três fases para uma sessão de comunicação entre um cliente (Bob) e um servidor (Alice), o qual possui um par de chaves pública/privada e um certificado que associa sua identidade à chave pública.

Apresentação (*handshake*)

Durante a fase de apresentação, Bob precisa (a) estabelecer uma conexão TCP com Alice, (b) verificar se ela é *realmente* Alice, e (c) enviar-lhe uma chave secreta mestre, que será utilizada por Alice e Bob para criar todas as chaves simétricas de que eles precisam para a sessão SSL. Essas três etapas estão ilustradas na Figura 8.25. Observe que, uma vez a conexão TCP sendo estabelecida, Bob envia a Alice uma mensagem "*hello*". Alice, então, responde com seu certificado, que contém sua chave pública. Conforme discutido na Seção 8.3, como o certificado foi assinado por uma CA, Bob sabe, com certeza, que a chave pública no certificado pertence a Alice. Ele então cria um Segredo Mestre (MS) (que será usado somente para esta sessão SSL), codifica o MS com a chave pública de Alice para criar o Segredo Mestre Cifrado (EMS), enviando-o para Alice, que o decodifica com sua chave privada para obter o MS. Após essa fase, Bob e Alice (e mais ninguém) sabem o segredo mestre para esta sessão SSL.

Derivação de chave

A princípio, o MS, agora compartilhado por Bob e Alice, poderia ser usado como a chave de sessão simétrica para toda a verificação subsequente de criptografia e integridade dos dados. Entretanto, em geral é considerado mais seguro para Alice e Bob usarem, individualmente, chaves criptográficas diferentes, bem como chaves diferentes para criptografia e verificação da integridade. Assim, Alice e Bob usam o MS para criar quatro chaves:

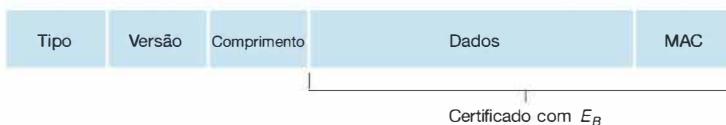
- E_B = chave de criptografia de sessão para dados enviados de Bob para Alice
- M_B = chave MAC de sessão para dados enviados de Bob para Alice
- E_A = chave de criptografia de sessão para dados enviados de Alice para Bob
- M_A = chave MAC de sessão para dados enviados de Alice para Bob

FIGURA 8.25 A APRESENTAÇÃO QUASE-SSL, INICIANDO COM UMA CONEXÃO TCP

Alice e Bob podem criar quatro chaves a partir do MS. Isso poderia ser feito apenas dividindo o MS em quatro chaves. (Mas, no SSL real, é um pouco mais complicado, como veremos.) Ao final da fase de derivação de chave, Alice e Bob têm todas as quatro chaves. As duas de criptografia serão usadas para cifrar dados; as duas chaves MAC serão usadas para verificar a integridade dos dados.

Transferência de dados

Agora que Alice e Bob compartilham as mesmas quatro chaves de sessão (E_B, M_B, E_A e M_A), podem começar a enviar dados protegidos um ao outro por meio da conexão TCP. Como o TCP é um protocolo de fluxo de bytes, uma abordagem natural seria o SSL cifrar dados da aplicação enquanto são enviados e, então, transmitir os dados cifrados para o TCP. Mas, se fizéssemos isso, onde colocaríamos o MAC para a verificação da integridade? Decerto não queremos esperar até o fim da sessão TCP para verificar a integridade de todos os dados de Bob que foram enviados por toda a sessão! Para abordar essa questão, o SSL divide o fluxo de dados em *registros*, anexa um MAC a cada registro para a verificação da integridade, e cifra o registro+MAC. Para criar o MAC, Bob insere os dados de registro junto com a chave M_B em uma função de *hash*, conforme discutido na Seção 8.3. Para cifrar o pacote registro+MAC, Bob usa sua chave de criptografia de sessão E_B. Esse pacote cifrado é então transmitido ao TCP para o transporte pela Internet.

Embora essa abordagem vá longe, o fornecimento de integridade dos dados para um fluxo inteiro de mensagem ainda não é à prova de falhas. Em particular, suponha que Trudy seja uma mulher no meio e possua a habilidade de inserir, excluir e substituir segmentos no fluxo dos segmentos TCP enviados entre Alice e Bob. Trudy, por exemplo, poderia capturar dois segmentos enviados por Bob, inverter sua ordem, ajustar os números de sequência TCP (que não estão cifrados) e enviar dois segmentos na ordem inversa para Alice. Supondo que cada segmento TCP encapsula exatamente um registro, vamos verificar como Alice os processaria.

1. O TCP que está sendo executado em Alice pensaria que está tudo bem e passaria os dois registros para a subcamada SSL.
2. O SSL em Alice decodificaria os dois registros.
3. O SSL em Alice usaria o MAC em cada registro para verificar a integridade dos dados dos dois registros.
4. O SSL passaria, então, os fluxos de bytes decifrados dos dois registros à camada de aplicação; mas o fluxo de bytes completo recebido por Alice não estaria na ordem correta por causa da inversão dos registros!

Recomendamos que você analise cenários diferentes para quando Trudy remove segmentos e quando Trudy repete segmentos.

A solução para esse problema, como você deve ter imaginado, é usar números de sequência. O SSL os utiliza da seguinte maneira. Bob mantém um contador de números de sequência, que se inicia no zero e vai aumentando para cada registro SSL que ele envia. Bob, na verdade, não inclui um número de sequência no próprio registro, mas no cálculo do MAC. Desse modo, o MAC é agora um *hash* dos dados mais uma chave MAC M_B *mais o número de sequência atual.* Alice rastreia os números de sequência de Bob, permitindo que ela verifique a integridade dos dados de um registro incluindo o número de sequência apropriado no cálculo do MAC. O uso de números de sequência SSL impede que Trudy realize um ataque da mulher do meio, como reordenar ou repetir os segmentos. (Por quê?)

Registro SSL

O registro SSL (assim como o registro quase-SSL) é ilustrado na Figura 8.26. Ele consiste em um campo de tipo, um campo de versão, um campo de comprimento, um campo de dados e um campo MAC. Observe que os primeiros três campos não estão cifrados. O campo de tipo indica se o registro é uma mensagem de apresentação ou uma mensagem que contém dados da aplicação. É também usado para encerrar a conexão SSL, como discutido a seguir. Na extremidade receptora, o SSL usa o campo de comprimento para extrair os registros SSL do fluxo de bytes TCP da entrada. O campo de versão não precisa de explicações.

FIGURA 8.26 FORMATO DE REGISTRO PARA O SSL

8.6.2 Uma visão mais completa

A subseção anterior abordou o protocolo quase-SSL; serviu para nos dar uma compreensão básica de por que e como funciona o SSL. Agora que temos essa compreensão básica sobre o SSL, podemos nos aprofundar mais e examinar os princípios básicos do verdadeiro protocolo SSL. Além da leitura desta descrição, recomendamos que você complete o laboratório Wireshark SSL, disponível no site de apoio deste livro.

Apresentação SSL

O SSL não exige que Alice e Bob usem um algoritmo específico de chave simétrica, um algoritmo de chave pública ou um MAC específico. Em vez disso, permite que Alice e Bob combinem os algoritmos criptográficos no início da sessão SSL, durante a fase de apresentação. Ademais, durante essa fase, Alice e Bob enviam *nonces* um ao outro, que são usados na criação de chaves de sessão (E_B, M_B, E_A e M_A), As etapas da apresentação do SSL real são:

1. O cliente envia uma lista de algoritmos criptográficos que ele suporta, junto com um *nonce* do cliente.
2. A partir da lista, o servidor escolhe um algoritmo simétrico (por exemplo, AES), um algoritmo de chave pública (por exemplo, RSA com um comprimento de chave específico) e um algoritmo MAC. Ele devolve ao cliente suas escolhas, bem como um certificado e um *nonce* do servidor.
3. O cliente verifica o certificado, extrai a chave pública do servidor, gera um Segredo Pré-Mestre (PMS), cifra o PMS com a chave pública do servidor e envia o PMS cifrado ao servidor.
4. Utilizando a mesma função de derivação de chave (conforme especificado pelo padrão SSL), o cliente e o servidor calculam independentemente o Segredo Mestre (MS) do PMS e dos *nonces*. O MS é então dividido para gerar as duas chaves de criptografia e duas chaves MAC. Além disso, quando a cifra simétrica selecionada emprega o CBC (como 3DES ou AES), então dois Vetores de Inicialização (IVs) — um para cada lado da conexão — são também obtidos do MS. De agora em diante, todas as mensagens enviadas entre o cliente e o servidor são cifradas e autenticadas (com o MAC).
5. O cliente envia um MAC de todas as mensagens de apresentação.
6. O servidor envia um MAC de todas as mensagens de apresentação.

As duas últimas etapas protegem a apresentação da adulteração. Para entender, observe que no passo 1 o cliente normalmente oferece uma lista de algoritmos — alguns fortes e outros fracos. Essa lista é enviada em texto aberto, visto que os algoritmos de criptografia e chaves ainda não foram consentidos. Trudy, como uma mulher no meio, poderia excluir os algoritmos mais fortes da lista, obrigando o cliente a selecionar um algoritmo fraco. Para evitar o ataque de adulteração, na etapa 5 o cliente envia um MAC da concatenação de todas as mensagens de apresentação que ele enviou e recebeu. O servidor pode comparar esse MAC com o das mensagens de apresentação que ele enviou e recebeu. Se houver uma inconsistência, o servidor pode finalizar a conexão. De maneira semelhante, o servidor envia um MAC das mensagens de apresentação que encontrou, permitindo que o cliente verifique a presença de inconsistências.

Você talvez esteja questionando por que existem *nonces* nas etapas 1 e 2. Os números de sequência não são suficientes para prevenir o ataque de repetição de segmento? A resposta é sim, mas eles não previnem sozinhos "o ataque de repetição de conexão". Considere o seguinte ataque de repetição de conexão. Suponha que Trudy analise todas as mensagens entre Alice e Bob. No dia seguinte, ela se passa por Bob e envia a Alice exatamente a

mesma sequência de mensagens que Bob enviou a Alice no dia anterior. Se Alice não usar os *nonces*, ela responderá exatamente com a mesma sequência de mensagens que enviou no dia anterior. Alice não suspeitará de nada estranho, pois cada mensagem que ela recebe passará pela verificação de integridade. Se Alice for um servidor de comércio eletrônico, pensará que Bob está efetuando um segundo pedido (para a mesma coisa). Por outro lado, ao incluir um *nonce* no protocolo, Alice enviará *nonces* diferentes para cada sessão TCP, fazendo que as chaves de criptografia sejam diferentes nos dois dias. Portanto, quando Alice recebe os registros SSL repetidos de Trudy, eles falharão na verificação de integridade, e a transação do site de comércio eletrônico falso não acontecerá. Em resumo, no SSL, os *nonces* são usados para proteger o "ataque de repetição de conexão", e os números de sequência são usados para defender a repetição de pacotes individuais durante uma sessão em andamento.

Encerramento de conexão

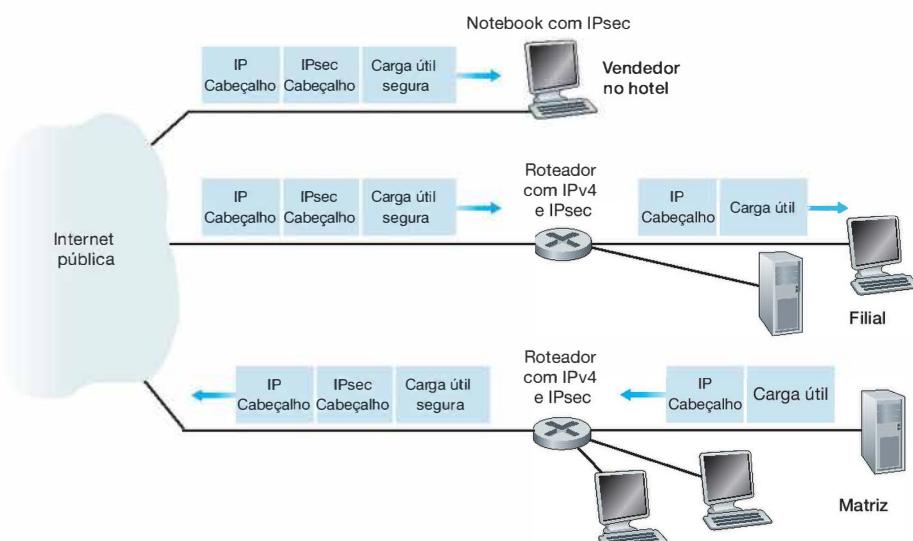
Em algum momento, Bob ou Alice desejarão finalizar a sessão SSL. Um método seria deixar Bob finalizar a sessão SSL apenas encerrando a conexão TCP subjacente — ou seja, enviando um segmento TCP FIN para Alice. Mas esse plano ingênuo prepara o terreno para o *ataque por truncamento* pelo qual Trudy, mais uma vez, se localiza no meio de uma sessão SSL em andamento e a finaliza antes da hora com um TCP FIN. Se Trudy fizesse isso, Alice acharia que recebeu todos os dados de Bob quando, na verdade, recebeu só uma parte deles. A solução para esse problema é indicar, no campo de tipo, se o registro serve para encerrar a sessão SSL. (Embora o tipo SSL seja enviado em aberto, ele é autenticado no receptor usando o MAC do registro.) Ao incluir esse campo, se Alice fosse receber um TCP FIN antes de receber um registro SSL de encerramento, ela saberia que algo estranho estava acontecendo.

Isso conclui nossa introdução ao SSL. Vimos que ele usa muitos dos princípios da criptografia discutidos nas seções 8.2 e 8.3. Os leitores que querem explorar o SSL mais profundamente podem consultar o livro de Rescorla sobre o SSL [Rescorla, 2001].

8.7 SEGURANÇA NA CAMADA DE REDE: IPSEC E REDES VIRTUAIS PRIVADAS

O protocolo IP de segurança, mais conhecido como **IPsec**, provê segurança na camada de rede. O IPsec protege os datagramas IP entre quaisquer entidades da camada de rede, incluindo hospedeiros e roteadores. Como descreveremos em breve, muitas instituições (corporações, órgãos do governo, organizações sem fins lucrativos etc.) usam o IPsec para criar **redes virtuais privadas (VPNs)** que trabalham em cima da Internet pública.

Antes de falar sobre o IPsec, vamos voltar e considerar o que significa prover sigilo na camada de rede. Com o sigilo da camada de rede entre um par de entidades da rede (por exemplo, entre dois roteadores, entre dois hospedeiros, ou entre um roteador e um hospedeiro), a entidade remetente cifra as cargas úteis de todos os datagramas que envia à entidade destinatária. A carga útil cifrada poderia ser um segmento TCP, um segmento UDP e uma mensagem ICMP etc. Se esse serviço estivesse em funcionamento, todos os dados enviados de uma entidade a outra — incluindo e-mail, páginas Web, mensagens de apresentação TCP e mensagens de gerenciamento (como ICMP e SNMP) — ficariam ocultos de qualquer intruso que estivesse investigando a rede. Por essa razão, a segurança na camada de rede é conhecida por prover "cobertura total".

Além do sigilo, um protocolo de segurança da camada de rede poderia prover outros serviços de segurança. Por exemplo, fornecer autenticação da origem, de modo que a entidade destinatária possa verificar a origem do datagrama protegido. Um protocolo de segurança da camada de rede poderia oferecer integridade dos dados, para que a entidade destinatária verificasse qualquer adulteração do datagrama que pudesse ter ocorrido enquanto o datagrama estava em trânsito. Um serviço de segurança da camada de rede também poderia oferecer a prevenção do ataque de repetição, querendo dizer que Bob conseguiria detectar quaisquer datagramas duplicados que um atacante pudesse inserir. Veremos em breve que o IPsec provê mecanismos para todos esses serviços de segurança, ou seja, para sigilo, autenticação da origem, integridade dos dados e prevenção do ataque de repetição.

8.7.1 IPsec e redes virtuais privadas (VPNs)

Uma instituição que se estende por diversas regiões geográficas muitas vezes deseja ter sua própria rede IP, para que seus hospedeiros e servidores consigam enviar dados um ao outro de uma maneira segura e sigilosa. Para alcançar essa meta, a instituição poderia, na verdade, empregar uma rede física independente — incluindo roteadores, enlaces e uma infraestrutura de DNS — que é completamente separada da Internet pública. Essa rede disjunta, reservada a uma instituição particular, é chamada de **rede privada**. Como é de se esperar, uma rede privada pode ser muito cara, já que a instituição precisa comprar, instalar e manter sua própria infraestrutura de rede física.

Em vez de implementar e manter uma rede privada, hoje muitas instituições criam VPNs em cima da Internet pública. Com uma VPN, o tráfego interdepartamental é enviado por meio da Internet pública e não de uma rede fisicamente independente. Mas, para prover sigilo, esse tráfego é criptografado antes de entrar na Internet pública. Um exemplo simples de VPN é mostrado na Figura 8.27. Aqui, a instituição consiste em uma matriz, uma filial e vendedores viajantes, que normalmente acessam a Internet do seu quarto de hotel. (A figura mostra só um vendedor.) Nesta VPN, quando dois hospedeiros dentro da matriz enviam datagramas IP um ao outro ou quando dois hospedeiros dentro de uma filial querem se comunicar, eles usam o bom e velho IPv4 (ou seja, sem os serviços IPsec). Entretanto, quando dois dos hospedeiros da instituição se comunicam por um caminho que cruza a Internet pública, o tráfego é codificado antes de entrar na Internet.

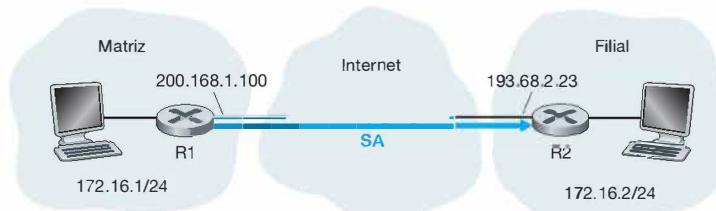
Para entender como a VPN funciona, vamos examinar um exemplo simples no contexto da Figura 8.27. Quando um hospedeiro na matriz envia um datagrama IP a um vendedor no hotel, o roteador de borda na matriz converte o datagrama IPv4 em um IPsec e o encaminha à Internet. Esse datagrama IPsec, na verdade, possui um cabeçalho IPv4 tradicional, de modo que os roteadores na Internet pública processam o datagrama como se ele fosse um IPv4 comum — para eles, o datagrama é perfeitamente comum. Mas, conforme ilustrado na Figura 8.27, a carga útil do datagrama IPsec inclui um cabeçalho IPsec, que é utilizado para o processamento do IPsec; além disso, a carga útil do datagrama IPsec está codificada. Quando o datagrama IPsec chegar ao notebook do vendedor, o sistema operacional no notebook decodifica a carga útil (e provê outros serviços de segurança, como a verificação da integridade dos dados) e passa a carga útil não codificada para o protocolo da camada superior (por exemplo, para o TCP ou UDP).

FIGURA 8.27 REDE VIRTUAL PRIVADA (VPN)

Acabamos de dar uma visão geral de alto nível de como uma instituição pode implementar o IPsec para criar uma VPN. Para ver a floresta por entre as árvores, deixamos de lado muitos detalhes importantes. Vamos agora dar uma olhada neles mais de perto.

8.7.2 Os protocolos AH e ESP

O IPsec é um material um tanto complexo — ele é definido em mais de uma dúzia de RFCs. Dois RFCs importantes são RFC 4301, o qual descreve a arquitetura de segurança IP geral, e RFC 6071, que fornece uma visão geral dos protocolos IPsec. Nossa meta neste livro, como de costume, não é apenas reprocessar os RFCs secos e complexos, mas fazer uma abordagem mais operacional e pedagógica para descrever os protocolos.

No conjunto dos protocolos IPsec, existem dois principais: o protocolo **Cabeçalho de Autenticação (AH)** e o protocolo **Carga de Segurança de Encapsulamento (ESP)**. Quando uma entidade remetente IPsec (em geral um hospedeiro ou um roteador) envia datagramas seguros a uma entidade destinatária (também um hospedeiro ou um roteador), ela utiliza o protocolo AH ou o protocolo ESP. O protocolo AH provê autenticação da origem e integridade dos dados, mas *não* provê sigilo. O protocolo ESP provê autenticação da origem, integridade dos dados *e* sigilo. Visto que o sigilo é muitas vezes essencial às VPNs e outras aplicações IPsec, o protocolo ESP é muito mais utilizado do que o protocolo AH. Para desmistificar o IPsec e evitar suas complexidades, a partir de agora estaremos focados exclusivamente no protocolo ESP. Os leitores que desejam aprender também sobre o protocolo AH devem explorar os RFCs e outros recursos on-line.

8.7.3 Associações de segurança

Os datagramas IPsec são enviados entre pares de entidades da rede, tal como entre hospedeiros, entre dois roteadores, ou entre um hospedeiro e um roteador. Antes de enviar datagramas IPsec da entidade remetente à destinatária, essas entidades criam uma conexão lógica da camada de rede, denominada **associação de segurança (SA)**. Uma SA é uma conexão lógica simples; ou seja, ela é unidirecional do remetente ao destinatário. Se as duas entidades querem enviar datagramas seguros entre si, então duas SAs (ou seja, duas conexões lógicas) precisam ser estabelecidas, uma em cada direção.

Por exemplo, considere mais uma vez uma VPN institucional na Figura 8.27. Essa instituição consiste em uma filial, uma matriz e, digamos, n vendedores viajantes. Por exemplo, vamos supor que haja tráfego IPsec bidirecional entre a matriz e a filial e tráfego IPsec bidirecional entre a matriz e os vendedores viajantes. Quantas SAs existem nessa VPN? Para responder a essa questão, observe que há duas SAs entre o roteador de borda da matriz e o de borda da filial (uma em cada direção); para cada notebook do vendedor, há duas SAs entre o roteador de borda da matriz e o notebook (de novo, uma em cada direção). Portanto, no total, há $(2 + 2n)$ SAs. *Mantenha em mente, entretanto, que nem todo o tráfego enviado à Internet pelos roteadores de borda ou pelos notebooks será protegido por IPsec.* Podemos citar como exemplo um hospedeiro na matriz que quer acessar ao servidor Web (como Amazon ou Google) na Internet pública. Assim, o roteador de borda (e os notebooks) emitirão na Internet tanto datagramas IPv4 comuns como datagramas IPsec seguros.

Vamos agora olhar "por dentro" de uma SA. Para tornar essa discussão tangível e concreta, utilizaremos o contexto de uma SA do roteador R1 ao roteador R2 na Figura 8.28. (Você pode pensar no Roteador R1 como o roteador de borda da matriz, e o Roteador R2 como o roteador de borda da filial, conforme a Figura 8.27.) O Roteador R1 manterá as informações de estado sobre essa SA, as quais incluirão:

- Um identificador de 32 bits para a SA, denominado **Índice de Parâmetro de Segurança (SPI)**
- A interface remetente da SA (neste caso 200.168.1.100) e a interface destinatária da SA (neste caso 193.68.2.23)
- O tipo de criptografia a ser usada (por exemplo, 3DES com CBC)
- A chave de criptografia

FIGURA 8.28 ASSOCIAÇÃO DE SEGURANÇA (SA) DE R1 A R2

- O tipo de verificação de integridade (por exemplo, HMAC com MD5)
- A chave de autenticação

Quando o roteador R1 precisa construir um datagrama IPsec para encaminhar por essa SA, ele acessa as informações de estado para determinar como deveria autenticar e criptografar o datagrama. De maneira semelhante, o roteador R2 manterá as mesmas informações de estado sobre essa SA e as usará para autenticar e decodificar qualquer datagrama IPsec que chegar da SA.

Uma entidade IPsec (roteador ou hospedeiro) muitas vezes guarda essas informações de estado para muitas SAs. No exemplo sobre a VPN na Figura 8.27 com *n* vendedores, o roteador de borda guarda informações de estado para (2 + 2*n*) SAs. Uma entidade IPsec armazena as informações de estado para todas as suas SAs em seu **Banco de Dados de Associação de Segurança (SAD),** o qual é uma estrutura de dados do núcleo do sistema operacional.

8.7.4 O datagrama IPsec

Após descrever sobre as SAs, podemos agora descrever o verdadeiro datagrama IPsec. Ele possui duas formas diferentes de pacotes, uma para o **modo túnel** e outra para o **modo transporte**. O modo túnel, o mais apropriado para as VPNs, é mais implementado do que o modo transporte. Para desmistificar o IPsec e evitar suas complexidades, a partir de agora vamos nos concentrar exclusivamente no modo túnel. Uma vez que você tiver uma compreensão sólida do modo túnel, poderá com facilidade aprender, por sua conta, o modo transporte.

O formato do pacote do datagrama IPsec é ilustrado na Figura 8.29. Você pode pensar que os formatos do pacote são maçantes e sem graça, mas logo veremos que o datagrama IPsec na verdade parece e tem gosto de uma iguaria mexicana! Vamos examinar os campos do IPsec no contexto da Figura 8.28. Suponha que o roteador R1 receba um datagrama IPv4 comum do hospedeiro 172.16.1.17 (na rede da matriz) destinado ao hospedeiro 172.16.2.48 (na rede da filial). R1 utiliza a seguinte receita para converter esse "datagrama IPv4 original" em um datagrama IPsec:

- Anexa ao final do datagrama IPv4 original (que inclui os campos de cabeçalho originais) um campo de "*trailer* ESP"

FIGURA 8.29 FORMATO DO DATAGRAMA IPSEC

- Codifica o resultado utilizando o algoritmo e a chave especificados pela SA
- Anexa na frente dessa quantidade codificada um campo chamado "cabeçalho ESP"; o pacote resultante é chamado de "*enchilada*"
- Cria uma autenticação MAC sobre a *enchilada inteira*, usando o algoritmo e a chave especificados na SA
- Anexa a MAC atrás da *enchilada*, formando a *carga útil*
- Por fim, cria um cabeçalho IP novo com todos os campos de cabeçalho IPv4 clássicos (juntos, normalmente com 20 bytes de comprimento), o qual se anexa antes da carga útil

Observe que o datagrama IP resultante é um autêntico datagrama IPv4, com os tradicionais campos de cabeçalho IPv4 acompanhados por uma carga útil. Mas, neste caso, a carga útil contém um cabeçalho ESP, o datagrama IP original, um trailer ESP e um campo de autenticação ESP (com o datagrama original e o trailer ESP cifrados). O datagrama IP original possui o endereço IP remetente 172.16.1.17 e o endereço IP destinatário 172.16.2.48. Em razão de o datagrama IPsec incluir o IP original, esses endereços são inclusos (e cifrados) como parte da carga útil do pacote IPsec. Mas e os endereços IP remetente e destinatário que estão no novo cabeçalho IP, ou seja, o cabeçalho localizado à esquerda do datagrama IPsec? Como você pode esperar, eles são estabelecidos para as interfaces do roteador remetente e destinatário nas duas extremidades dos túneis, isto é, 200.168.1.100 e 193.68.2.23. Além disso, o número do protocolo nesse novo campo de cabeçalho IPv4 não será o número do TCP, UDP ou SMTP, mas igual a 50, determinando que esse é um datagrama IPsec que está utilizando o protocolo ESP.

Após R1 enviar um datagrama IPsec à Internet pública, ele passará por diversos roteadores antes de chegar ao R2. Cada um desses roteadores processará o datagrama como se fosse um datagrama comum — eles são inconscientes do fato de que o datagrama está transportando dados cifrados pelo IPsec. Para esses roteadores da Internet pública, como o endereço IP remetente no cabeçalho externo é R2, o destino final do datagrama é R2.

Depois de acompanhar um exemplo de como o datagrama IPsec é construído, vamos olhar de perto os ingredientes da *enchilada*. Vemos na Figura 8.29 que o trailer ESP consiste em três campos: enchimento, tamanho do enchimento e próximo cabeçalho. Lembre-se de que cifras de bloco exigem que a mensagem a ser cifrada seja um múltiplo inteiro do comprimento de bloco. O enchimento (que consiste em bytes sem significado) é usado de modo que, quando adicionada ao datagrama original (junto com o tamanho do enchimento e próximo cabeçalho), a "mensagem" resultante tenha um número inteiro de blocos. O campo tamanho do enchimento indica à entidade destinatária quanto enchimento foi inserido (e, portanto, precisa ser removido). O próximo cabeçalho identifica o tipo (por exemplo, UDP) de dados contidos no campo de dados da carga útil. Os dados da carga útil (em geral o datagrama IP original) e o trailer ESP são concatenados e, então, cifrados.

Anexado na frente dessa unidade cifrada está o cabeçalho ESP, o qual é enviado em aberto e consiste em dois campos: o SPI e o campo de número de sequência. O SPI indica à entidade destinatária a SA à qual o datagrama pertence; essa entidade pode, então, indexar seu SAD com o SPI para determinar os algoritmos e chaves apropriados de autenticação/decriptação. O campo de número de sequência é usado para a proteção contra ataques de repetição.

A entidade remetente também anexa uma autenticação MAC. Como já dissemos, a entidade remetente calcula um MAC por toda a enchilada (que consiste em um cabeçalho ESP, o datagrama IP original e o trailer ESP — com a criptografia do datagrama e do trailer). Lembre-se de que, para calcular um MAC, o remetente anexa uma chave MAC secreta à enchilada e calcula um *hash* de tamanho fixo do resultado.

Quando R2 recebe o datagrama IPsec, ele observa que o endereço IP destinatário do datagrama é o próprio R2, que, então, processa o datagrama. Como o campo de protocolo (no cabeçalho IP à esquerda) é 50, R2 entende que deve aplicar o processamento IPsec ESP ao datagrama. Primeiro, alinhando na *enchilada*, E2 usa o SPI para determinar a qual SA o datagrama pertence. Segundo, ele calcula o MAC da *enchilada* e verifica se o MAC é compatível com o valor do campo MAC ESP. Se for, ele sabe que a *enchilada* vem de R1 e que não foi adulterada. Terceiro, verifica o campo número de sequência para ver se o datagrama é novo (e não repetido). Quarto, ele decodifica a unidade cifrada utilizando o algoritmo e a chave de criptografia associados à SA. Quinto, ele remove o enchimento e extrai o datagrama IP original básico. E, por fim, sexto, ele encaminha o datagrama original à rede da filial em direção a seu destino final. Ufa, que receita complicada, não é? Ninguém disse que era fácil preparar e desvendar uma *enchilada*!

Na verdade, existe outra importante sutileza que precisa ser abordada. Ela se baseia na seguinte questão: Quando R1 recebe um datagrama (inseguro) de um hospedeiro na rede da matriz, e esse datagrama é destinado a algum endereço IP destinatário fora da matriz, como R1 sabe se ele deve ser convertido em um datagrama IPsec? E se ele vai ser processado por um IPsec, como R1 sabe qual SA (de muitas SAs em seu SAD) deve ser usada para construir o datagrama IPsec? O problema é resolvido da seguinte maneira. Junto com um SAD, a entidade IPsec também mantém outra estrutura de dados denominada **Banco de Dados de Política de Segurança (SPD)**. Este indica que tipos de datagramas (como uma função do endereço IP remetente, endereço IP destinatário e tipo do protocolo) serão processados pelo IPsec; e para aqueles que serão processados pelo IPsec, qual SA deve ser usada. De certa forma, as informações em um SPD indicam "o que" fazer com um datagrama que está chegando; as informações no SAD indicam "como" fazer isso.

Resumo dos serviços IPsec

Quais serviços o IPsec provê, exatamente? Vamos examinar tais serviços do ponto de vista de um atacante, digamos Trudy, que é uma mulher no meio, situada entre R1 e R2 na Figura 8.28. Suponha por toda essa discussão que Trudy não conhece as chaves de autenticação e criptografia usadas pela SA. O que Trudy pode e não pode fazer? Primeiro, ela não pode ver o datagrama original. Na verdade, não só os dados do datagrama original estão ocultos de Trudy como também o número de protocolo, o endereço IP remetente e o endereço IP destinatário. Em relação aos datagramas enviados através da SA, Trudy sabe apenas que eles vieram de algum hospedeiro em 172. 16.1.0/24 e são destinados a algum hospedeiro em 172.16.2.0/24. Ela não sabe se está carregando dados TCP, UDP ou ICMP; ela não sabe que está carregando HTTP, SMTP ou quaisquer outros tipos de dados de aplicação. Esse sigilo, portanto, vai além do SSL. Segundo, suponha que Trudy tente adulterar um datagrama na SA alterando alguns de seus bits. Quando o datagrama alterado chegar a R2, a verificação de integridade falhará (usando um MAC), impedindo a tentativa maliciosa de Trudy mais uma vez. Terceiro, imagine que Trudy tente se passar por R1, criando um datagrama IPsec com remetente 200.168.1.100 e destinatário 193.68.2.23. O ataque será inútil, pois a verificação da integridade do datagrama em R2 falhará novamente. Por fim, em razão de o IPsec incluir números de sequência, Trudy não poderá criar um ataque de repetição bem-sucedido. Em resumo, conforme afirmado no início desta seção, o IPsec oferece — entre qualquer par de dispositivos que processam pacotes através da camada de rede — sigilo, autenticação da origem, integridade dos dados e proteção contra ataque de repetição.

8.7.5 IKE: Gerenciamento de chave no IPsec

Quando uma VPN possui um número pequeno de pontos finais (por exemplo, apenas dois roteadores como na Figura 8.28), o administrador de rede pode inserir manualmente informações sobre a SA (algoritmos e chaves de criptografia/autenticação e os SPIs) nos SADs dos pontos de chegada. Essa "teclagem manual" é claramente impraticável para uma VPN grande, a qual pode consistir em centenas ou mesmo milhares de roteadores e hospedeiros IPsec. Implementações grandes e geograficamente distribuídas exigem um mecanismo automático para a criação das SAs. O IPsec o faz com o protocolo de Troca de Chave (IKE), especificado em RFC 5996.

O IKE tem semelhanças com a apresentação (*handshake*) em SSL (consulte a Seção 8.6). Cada entidade IPsec possui um certificado, o qual inclui a chave pública da entidade. Da mesma forma que o SSL, o protocolo IKE tem os dois certificados de troca de entidades, autenticação de negociação e algoritmos de criptografia, e troca informações de chave com segurança para criar chaves de sessão nas SAs IPsec. Diferente do SSL, o IKE emprega duas fases para realizar essas tarefas.

Vamos investigar essas duas fases no contexto de dois roteadores, R1 e R2, na Figura 8.28. A primeira fase consiste em duas trocas de pares de mensagem entre R1 e R2:

- Durante a primeira troca de mensagens, os dois lados usam Diffie-Hellman (consulte os Problemas no final do capítulo) para criar um **IKE SA** bidirecional entre os roteadores. Para nos confundir, esse IKE

SA bidirecional é totalmente diferente da SA IPsec discutida nas seções 8.6.3 e 8.6.4. O IKE SA provê um canal cifrado e autenticado entre os dois roteadores. Durante a primeira troca de par de mensagem, as chaves são estabelecidas para a criptografia e autenticação para o IKE SA. Também é estabelecido um segredo mestre que será usado para calcular chaves SA IPsec mais adiante na fase 2. Observe que, durante a primeira etapa, as chaves públicas e privadas RSA não são usadas. Em particular, nem R1 nem R2 revelam sua identidade assinando uma mensagem com sua chave privada.

- Durante a segunda troca de mensagens, ambos os lados revelam sua identidade assinando suas mensagens. Entretanto, as identidades não são reveladas a um analisador passivo, pois são enviadas por um canal seguro IKE SA. Também nessa fase, os dois lados negociam os algoritmos de autenticação e criptografia que serão empregados pelas SAs IPsec.

Na fase 2 do IKE, os dois lados criam uma SA em cada direção. Ao fim da fase 2, as chaves de sessão de criptografia e autenticação são estabelecidas em ambos os lados para as duas SAs. Eles podem, então, usar as SAs para enviar datagramas seguros, como descrito nas seções 8.6.3 e 8.6.4. A principal motivação de ter duas fases no IKE é o custo computacional — visto que a segunda fase não envolve qualquer chave pública de criptografia, o IKE pode criar um grande número de SAs entre as duas entidades IPsec com um custo computacional relativamente pequeno.

8.8 SEGURANÇA DE LANS SEM FIO

Segurança é uma preocupação importante em redes sem fio, em que as ondas de rádio carregando quadros podem se propagar muito além do prédio que contém os hospedeiros e as estações base sem fio. Nesta seção, apresentaremos uma breve introdução em segurança sem fio. Para obter mais informações, veja o livro de fácil leitura de Edney e Arbaugh [Edney, 2003].

O tema de segurança em 802.11 tem atraído muita atenção em círculos técnicos e na mídia. Apesar de discussões consideráveis terem sido feitas, poucos debates aconteceram — parece existir um entendimento universal de que a especificação 802.121 original contém uma série de falhas graves na segurança. De fato, podemos fazer o *download* de softwares de domínio público que tiram proveito dessas falhas, fazendo aqueles que usam mecanismos de segurança 802.11 comuns ficarem expostos a ataques de segurança tanto quanto os que não usam nenhum tipo de atributos de segurança.

Na seção seguinte, discutiremos os mecanismos de segurança inicialmente padronizados na especificação 802.11, conhecida como **Privacidade Equivalente Cabeada** (WEP, do inglês *Wired Equivalent Privacy*). Como o nome sugere, a WEP tem como propósito fornecer um nível de segurança semelhante ao que é encontrado em redes cabeadas. Então, discutiremos algumas falhas de segurança da WEP e examinaremos o padrão 802.11i, uma versão fundamentalmente mais segura do que 802.11, adotado em 2004.

8.8.1 Privacidade Equivalente Cabeada (WEP)

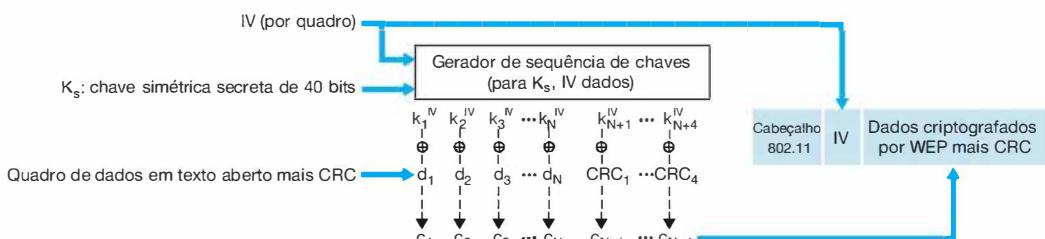
O protocolo IEEE 802.11 WEP foi criado em 2009 para fornecer autenticação e criptografia de dados entre um hospedeiro e um ponto de acesso sem fio (ou seja, a estação-base) usando uma técnica de chave compartilhada simétrica. A WEP não especifica um algoritmo de gerenciamento de chave, então supomos que o hospedeiro e o ponto de acesso sem fio de alguma forma concordam sobre a chave através de um método fora da banda. A autenticação é realizada da seguinte forma:

1. Um hospedeiro sem fio requisita uma autenticação por um ponto de acesso.
2. Um ponto de acesso responde ao pedido de autenticação com um valor de *nonce* de 128 bytes.
3. O hospedeiro sem fio criptografa o *nonce* usando uma chave simétrica que compartilha com o ponto de acesso.
4. O ponto de acesso decodifica o *nonce* criptografado do hospedeiro.

Se o *nonce* decodificado for compatível com o valor *nonce* originalmente enviado ao hospedeiro, então o hospedeiro é autenticado pelo ponto de acesso.

O algoritmo criptografado de dados WEP é ilustrado na Figura 8.30. Uma chave simétrica secreta de 40 bits, K_S, é presumidamente conhecida por ambos, hospedeiro e ponto de acesso. Além disso, um Vetor de Inicialização (IV, do inglês *Initialization Vector*) de 24 bits é anexado a uma chave de 40 bits para criar uma chave de 64 bits que serão usados para criptografar um único quadro. O IV mudará de um quadro para o outro e, por conseguinte, cada quadro será criptografado com uma chave de 64 bits diferente. A criptografia é efetuada da seguinte forma. Primeiro, um valor de 4 bytes de CRC (veja a Seção 5.2) é calculado para a carga útil de dados. Então, a carga útil e o CRC de quatro bytes são criptografados usando uma cifra de fluxo RC4. Não veremos os detalhes de um RC4 aqui (veja Schneier [1995] e Edney [2003] para obter mais detalhes). Para nossos objetivos, é suficiente saber que, quando o valor de uma chave (nesse caso, a chave de 64 bits (K_S, IV)) é apresentado ao algoritmo RC4, este produz um fluxo de valores de chave k_1^{IV}, k_2^{IV}, k_3^{IV},... que são usados para criptografar os dados e o valor CRC em um quadro. Para propósitos práticos, podemos pensar nessas operações como se fossem executadas em um byte por vez. A criptografia é realizada por OU-exclusivo no *i*-ésimo byte de dados, d_i, com a i-ésima chave, k_i^{IV}, no fluxo de valores de chaves produzido pelo par (K_s, IV) para produzir o *i*-ésimo byte do texto cifrado, c_i:

$$c_i = d_i \oplus k_i^{IV}$$

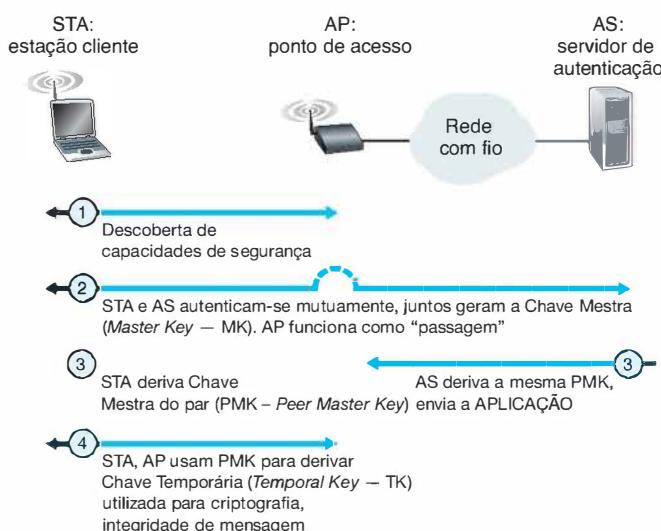
O valor de IV muda de um quadro ao próximo e é incluído *no texto aberto* do cabeçalho de cada quadro 802.11 criptografado por WEP, como ilustra a Figura 8.30. O receptor aceita a chave simétrica secreta de 40 bits que compartilha com o transmissor, anexa o IV e usa a chave de 64 bits resultante (que é idêntica à usada pelo transmissor para executar a criptografia) para decriptografar o quadro:

$$d_i = c_i \oplus k_i^{IV}$$

O uso adequado do algoritmo RC4 necessita que o mesmo valor da chave de 64 bits *nunca* seja usado mais de uma vez. Lembre-se de que a chave WEP muda em uma base quadro a quadro. Para determinado KS (que muda em raras ocasiões), isso significa que existem somente 2^{24} chaves únicas. Se estas são escolhidas de modo aleatório, podemos mostrar que a probabilidade de ter escolhido o mesmo valor de IV (e, portanto, usado a mesma chave de 64 bits) é de mais de 99% depois de somente 12.000 quadros [Walker, 2000; Edney, 2003]. Com um quadro de 1 Kbyte e a transmissão de dados de velocidade 11 Mbits/s, apenas alguns segundos são necessários antes que 12.000 quadros sejam transmitidos. Além do mais, já que IV é transmitido em texto aberto no quadro, um curioso saberá quando um valor IV duplicado for usado.

Para notar um dos vários problemas que ocorrem quando uma chave duplicada é usada, considere o seguinte ataque ao texto aberto escolhido por Trudy contra Alice. Suponha que Trudy (usando provavelmente uma falsificação de IP) envia uma solicitação a Alice (por exemplo, uma solicitação HTTP ou uma solicitação FTP) para transmitir um arquivo de conteúdo declarado, d_1, d_2, d_3, d_4... Trudy também nota os dados criptografados c_1, c_2, c_3, c_4... Já que $d_i = c_i \oplus k_i^{IV}$, se OU-exclusivo c_i com cada lado dessa igualdade, teremos

$$d_i \oplus c_i = k_i^{IV}$$

FIGURA 8.30 PROTOCOLO 802.11 WEP

Nessa relação, Trudy pode usar os valores conhecidos de d_i e c_i para calcular k_i^{IV}. Da próxima vez que Trudy vir o mesmo valor de IV sendo usado, ela saberá a sequência de chave $k_1^{IV}, k_2^{IV}, k_3^{IV},\ldots$ e assim será capaz de decriptografar a mensagem criptografada.

Existem outras diversas preocupações adicionais sobre a segurança da WEP. Fluhrer [2001] descreve um ataque aproveitando-se de uma fraqueza conhecida no RC4 quando certas chaves são escolhidas. Stubblefield [2002] discute maneiras eficazes de programar e tirar proveito desse ataque. Outra preocupação com a WEP envolve os bits CRC, mostrados na Figura 8.30, e transmitidos no quadro 802.11 para detectar bits alterados na carga útil. No entanto, um atacante que muda o conteúdo criptografado (por exemplo, substituindo textos ininteligíveis por dados criptografados originais), calcula o CRC através de textos ininteligíveis e coloca o CRC em um quadro WEP, pode produzir um quadro 802.11 que será aceito pelo receptor. O que é necessário nesse caso são técnicas de integridade de mensagem, como as estudadas na Seção 8.3 para detectar intercepções físicas ou substituições de conteúdo. Para obter mais informações sobre segurança da WEP, veja Edney [2003]; Walker [2000] Weatherspoon [2000] e suas referências a esse respeito.

8.8.2 IEEE 802.11i

Logo após o lançamento do IEEE 802.11 em 1999, começou o trabalho no desenvolvimento de uma versão nova e aprimorada do 802.11, com mecanismos de segurança mais fortes. O novo padrão, conhecido como 802.11i, passou por uma ratificação final em 2004. Como veremos, enquanto a WEP fornecia uma criptografia relativamente fraca, somente um meio de executar autenticação e nenhum mecanismo de distribuição de chaves, o IEEE 802.11i fornece formas de criptografia muito mais fortes, um conjunto extenso de mecanismos de autenticação e um mecanismo de distribuição de chaves. A seguir, apresentamos uma síntese do 802.11i; um excelente panorama técnico (em fluxo de áudio) sobre 802.11i é TechOnline [2012].

A Figura 8.31 dá uma visão geral sobre a estrutura do 802.11i. Além do cliente sem fio e do ponto de acesso, o 802.11i define um servidor de autenticação, com o qual o AP se comunica. Separar o servidor de comunicação do AP permite que um servidor de autenticação atenda a muitos APs, centralizando as decisões (muitas vezes

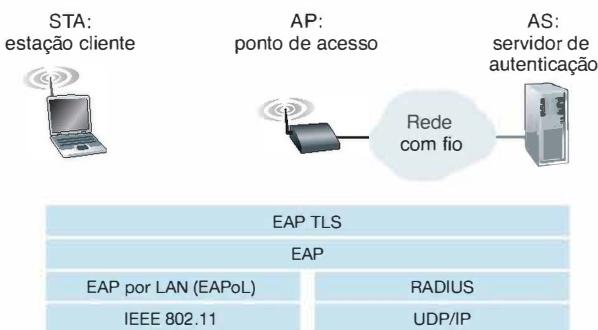



FIGURA 8.31 802.11i: QUATRO FASES DE OPERAÇÃO

confidenciais) relativas à autenticação e acesso em um único servidor, e mantendo os custos e a complexidade do AP menores. O 802.11i opera em quatro fases:

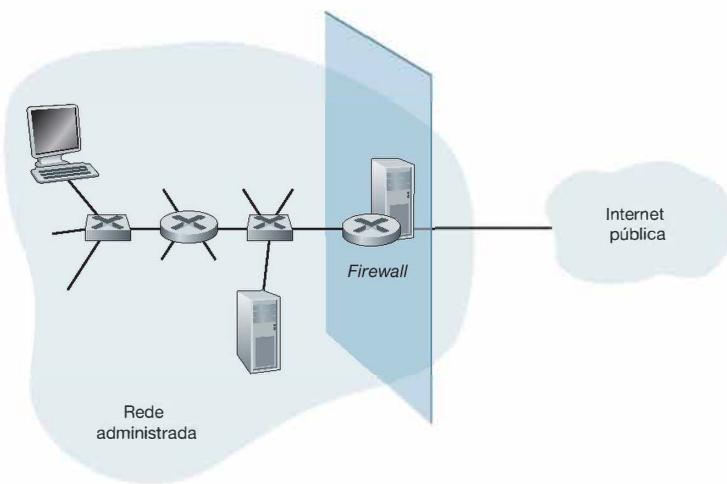
1. *Descoberta*. Na fase de descoberta, o AP anuncia sua presença e as formas de autenticação e criptografia que podem ser fornecidas ao nó do cliente sem fio. Então, o cliente solicita as formas específicas de autenticação e criptografia que deseja. Apesar de o cliente e o AP já estarem trocando mensagens, o cliente ainda não foi autenticado, nem tem uma chave criptografada, então vários passos ainda serão necessários antes que o cliente possa se comunicar com um hospedeiro remoto qualquer por um canal sem fio.
2. *Autenticação mútua e geração da Chave Mestra (MK)*. A autenticação ocorre entre o cliente sem fio e o servidor de autenticação. Nesta fase, o ponto de acesso age essencialmente como um repassador, encaminhando mensagens entre o cliente e o servidor de autenticação. O **Protocolo de Autenticação Extensível** (**EAP**, do inglês *Extensible Authentication Protocol*) [RFC 3748] define o formato da mensagem fim a fim usado em um modo simples de requisição/resposta de interação entre o cliente e o servidor de autenticação. Como mostrado na Figura 8.32, as mensagens EAP são encapsuladas usando um **EAPoL** (EAP através da LAN, [IEEE 802.1X]) e enviadas através de um enlace 802.11 sem fio. Então, estas mensagens EAP são desencapsuladas no ponto de acesso, e reencapsuladas usando um protocolo **RADIUS** para a transmissão por UDP/IP ao servidor de autenticação. Embora o protocolo e o servidor RADIUS [RFC 2865] não sejam requisitados pelo protocolo 802.11i, eles são componentes-padrão na prática para o 802.11i. O protocolo há pouco padronizado **DIAMETER** [RFC 3588] é propenso a substituir o RADIUS em um futuro próximo.
   Com o EAP, o servidor de autenticação pode escolher diversos modos para realizar a autenticação. Embora o 802.11i não exija um método específico de autenticação, o esquema de autenticação EAP-TLS [RFC 5216] muitas vezes é utilizado. O EAP-TLS usa técnicas de chaves públicas (incluindo a criptografia *nonce* e resumos de mensagens) semelhantes às que estudamos na Seção 8.3, que permitem que o cliente e o servidor de autenticação se autentiquem mutuamente, e produzam uma Chave Mestra (MK, do inglês *Master Key*) que é conhecida por ambas as partes.
3. *Geração de Chave Mestra Pareada (PMK)*. A MK é compartilhada secretamente apenas para o cliente e para o servidor de autenticação, sendo usada por eles para gerar uma segunda chave, a Chave Mestra Pareada (PMK, do inglês *Pairwise Master Key*). Então, o servidor de autenticação envia a PMK ao AP. E este é o ponto a que queríamos chegar! O cliente e o AP têm agora uma chave compartilhada (lembre-se de que em WEP o problema da distribuição de chaves não foi tratado) e se autenticaram mutuamente. Agora eles estão quase prontos para entrar em ação.

FIGURA 8.32 O EAP É UM PROTOCOLO FIM A FIM. AS MENSAGENS EAP SÃO ENCAPSULADAS USANDO UM EAPoL ATRAVÉS DE UM ENLACE SEM FIO ENTRE O CLIENTE E O PONTO DE ACESSO, E USANDO RADIUS ATRAVÉS DE UDP/IP ENTRE O PONTO DE ACESSO E O SERVIDOR DE AUTENTICAÇÃO

4. *Geração de Chave Temporal (TK).* Com a PMK, o cliente sem fio e o AP podem agora gerar chaves adicionais que serão usadas para comunicação. De interesse particular temos a Chave Temporal (TK, do inglês *Temporal Key*), que será usada para executar a criptografia dos dados enviados em nível de enlace pelo enlace sem fio e um hospedeiro remoto qualquer.

O 802.11i oferece várias formas de criptografia, incluindo um esquema de criptografia baseada em AES e uma versão reforçada da criptografia WEP.

8.9 SEGURANÇA OPERACIONAL: *FIREWALLS* E SISTEMAS DE DETECÇÃO DE INVASÃO

Vimos, em todo este capítulo, que a Internet não é um lugar muito seguro — os delinquentes estão por toda parte, criando todo tipo de destruição. Sabendo da natureza hostil da Internet, vamos considerar a rede de uma organização e um administrador de rede que a administra. Do ponto de vista de um administrador, o mundo está dividido claramente em dois campos — os mocinhos (que pertencem à organização que administra a rede e que deveriam poder acessar recursos dentro da rede que ele administra de um modo relativamente livre de restrições) e os bandidos (todo o resto, cujo acesso aos recursos da rede deve ser cuidadosamente inspecionado). Em muitas organizações, que vão de castelos medievais a modernos escritórios de empresas, há um único ponto de entrada/saída onde ambos, mocinhos e bandidos que entram e saem, passam por inspeção de segurança. Em castelos medievais, essa inspeção era feita em um portão, na extremidade de uma ponte levadiça; em escritórios empresariais, ela é feita na central de segurança. Em redes de computadores, quando o tráfego que entra/sai de uma rede passa por inspeção de segurança, é registrado, descartado ou transmitido; isso é feito por mecanismos operacionais conhecidos como *firewalls*, sistemas de detecção de invasão (IDSs) e sistemas de prevenção de invasão (IPSs).

8.9.1 *Firewalls*

Um ***firewall*** é uma combinação de hardware e software que isola a rede interna de uma organização da Internet em geral, permitindo que alguns pacotes passem e bloqueando outros. O *firewall* permite a um administrador de rede controlar o acesso entre o mundo externo e os recursos da rede que ele administra, gerenciando o fluxo de tráfego de e para esses recursos. Um *firewall* possui três objetivos:

- *Todo o tráfego de fora para dentro, e vice-versa, passa por um firewall.* A Figura 8.33 mostra um *firewall*, situado diretamente no limite entre a rede administrada e o resto da Internet. Embora grandes organizações possam usar diversos níveis de *firewalls* ou *firewalls* distribuídos [Skoudis, 2006], alocar um *firewall* em um único ponto de acesso à rede, conforme mostrado na Figura 8.33, facilita o gerenciamento e a execução de uma política de acesso seguro.
- *Somente o tráfego autorizado, como definido pela política de segurança local, poderá passar.* Com todo o tráfego que entra e sai da rede institucional passando pelo *firewall*, este pode limitar o acesso ao tráfego autorizado.
- *O próprio firewall é imune à penetração.* O próprio *firewall* é um mecanismo conectado à rede. Se não projetado ou instalado de modo adequado, pode ser comprometedor, oferecendo apenas uma falsa sensação de segurança (pior do que não ter nenhum *firewall*!).

Cisco e Check Point são dois dos principais fornecedores atuais de *firewall*. Você pode criar um *firewall* (filtro de pacotes) facilmente a partir de um sistema Linux usando *iptables* (software de domínio público que em geral acompanha o Linux).

Os *firewalls* podem ser classificados em três categorias: **filtros de pacotes tradicionais**, **filtros de estado** e **gateways de aplicação**. Abordaremos cada um nas subseções seguintes.

FIGURA 8.33 POSIÇÃO DO *FIREWALL* ENTRE A REDE ADMINISTRADA E O MUNDO EXTERIOR

Filtros de pacotes tradicionais

Como ilustra a Figura 8.33, uma organização normalmente tem um roteador de borda que conecta sua rede interna com seu ISP (e dali com a Internet pública, mais ampla). Todo o tráfego que sai ou que entra na rede interna passa por esse roteador e é nele que ocorre a **filtragem de pacotes**. Um filtro de pacotes examina cada datagrama que está sozinho, determinando se deve passar ou ficar baseado nas regras específicas definidas pelo administrador. As decisões de filtragem costumam ser baseadas em:

- Endereço IP de origem e de destino
- Tipo de protocolo no campo do datagrama IP: TCP, UDP, ICMP, OSPF etc.
- Porta TCP ou UDP de origem e de destino
- Bits de *flag* do TCP: SYN, ACK etc.
- Tipo de mensagem ICMP
- Regras diferentes para datagramas que entram e saem da rede
- Regras diferentes para diferentes interfaces do roteador

Um administrador de rede configura o *firewall* com base na política da organização. A política pode considerar a produtividade do usuário e o uso de largura de banda, bem como as preocupações com segurança da organização. A Tabela 8.5 lista diversas políticas que uma organização pode ter, e como elas seriam endereçadas com um filtro de pacotes. Por exemplo, se a organização não quer nenhuma conexão TCP de entrada, exceto aquelas para o servidor Web público, ela pode bloquear todos os segmentos TCP SYN de entrada, com exceção dos segmentos TCP SYN com porta de destino 80 e endereço IP de destino correspondente ao servidor Web. Se ela não quer que seus usuários monopolizem a largura de banda de acesso com aplicações de rádio via Internet, pode bloquear todo o tráfego UDP não importante (já que o rádio via Internet é em geral enviado por UDP). Se não quer que sua rede interna seja mapeada (por traceroute) por um estranho, pode bloquear todas as mensagens ICMP TTL expiradas que saem da rede da organização.

Uma política de filtragem também pode ser baseada na combinação de endereços e números de porta. Por exemplo, o roteador de filtragem poderia bloquear todos os datagramas Telnet (os que têm número de porta 23), exceto os que estão vindo ou indo de ou para uma lista de endereços IP específicos. Essa política permite conexões Telnet de e para os hospedeiros que estão na lista. Infelizmente, basear a política em endereços externos não oferece nenhuma proteção contra datagramas cujo endereço de origem foi falsificado.

TABELA 8.5 POLÍTICAS E REGRAS DE FILTRAGEM CORRESPONDENTES PARA UMA REDE DA ORGANIZAÇÃO 130.27/16 COM SERVIDOR WEB 130.207.244.203

Política	Configuração de *firewall*
Não há acesso exterior à Web	Descartar todos os pacotes de saída para qualquer endereço IP, porta 80
Não há conexões TCP de entrada, exceto aquelas apenas para o servidor Web público da organização	Descartar todos os pacotes TCP SYN para qualquer IP exceto 130.207.244.203, porta 80
Impedir que rádios Web devorem a largura de banda disponível	Descartar todos os pacotes UDP de entrada — exceto pacotes DNS
Impedir que sua rede seja usada por um ataque DoS *smurf*	Descartar todos os pacotes *ping* que estão indo para um endereço de difusão (por exemplo, 130.207.255.255)
Impedir que a rota de sua rede seja rastreada	Descartar todo o tráfego de saída ICMP com TTL expirado

A filtragem pode também ser baseada em o bit TCP ACK estar ou não marcado. Esse truque é bastante útil quando uma organização quer permitir que seus clientes internos se conectem com servidores externos, mas impedir que clientes externos se conectem com servidores internos. Lembre-se de que o primeiro segmento de todas as conexões TCP (veja a Seção 3.5) tem o bit ACK com valor 0, ao passo que todos os outros segmentos da conexão têm o bit ACK com valor 1. Assim, se uma organização quiser impedir que clientes externos iniciem uma conexão com servidores internos, ela apenas filtrará todos os segmentos que entram que tenham o bit ACK definido como 0. Essa política elimina todas as conexões TCP originadas do exterior, mas permite conexões que se originam internamente.

As regras do *firewall* são executadas em roteadores com listas de controle de acesso, tendo cada interface do roteador sua própria lista. Um exemplo de uma lista de controle de acesso para uma organização 222.22/16 é ilustrado na Tabela 8.6. Essa lista é para uma interface que conecta o roteador aos ISPs externos da organização. As regras são aplicadas a cada datagrama que atravessa a interface de cima para baixo. As primeiras duas regras juntas permitem que usuários internos naveguem na Web: a primeira permite que qualquer pacote TCP com porta de destino 80 saia da rede da organização; a segunda autoriza que qualquer pacote TCP com porta de origem 80 e o bit ACK marcado entrem na rede. Observe que se uma fonte externa tentar estabelecer uma conexão TCP com um hospedeiro interno, a conexão será bloqueada, mesmo que a porta de origem ou de destino seja 80. As duas regras juntas permitem que pacotes DNS entrem e saiam da rede da organização. Em resumo, essa lista de controle de acesso limitada bloqueia todo o tráfego exceto o tráfego Web iniciado de dentro da organização e do tráfego DNS. [CERT Filtering, 2012] apresenta uma lista de portas/protocolos para a filtragem de pacotes para evitar diversas brechas de segurança conhecidas nas aplicações de rede existentes.

TABELA 8.6 LISTA DE CONTROLE DE ACESSO PARA UMA INTERFACE DO ROTEADOR

Ação	Endereço de origem	Endereço de destino	Protocolo	Porta de origem	Porta de destino	Bit de flag
Permitir	222.22/16	Fora de 222.22/16	TCP	> 1023	80	Qualquer um
Permitir	Fora de 222.22/16	222.22/16	TCP	80	> 1023	ACK
Permitir	222.22/16	Fora de 222.22/16	UDP	> 1023	53	—
Permitir	Fora de 222.22/16	222.22/16	UDP	53	> 1023	—
Negar	Todos	Todos	Todos	Todos	Todos	Todos

Filtros de pacote com controle de estado

Em um filtro de pacotes tradicional, as decisões de filtragem são feitas em cada pacote isolado. Os filtros de estado rastreiam conexões TCP e usam esse conhecimento para tomar decisões sobre filtragem.

Para entender esses filtros de estado, vamos reexaminar a lista de controle de acesso da Tabela 8.6. Embora um tanto restritiva, essa lista, no entanto, permite que qualquer pacote que chegue de fora com um ACK = 1 e porta

de origem 80 atravesse o filtro. Esses pacotes poderiam ser usados em tentativas de destruir o sistema interno com pacotes defeituosos, realizar ataques de recusa de serviço, ou mapear a rede interna. A solução ingênua é bloquear pacotes TCP ACK também, mas tal método impediria que os usuários internos da organização navegassem na Web.

Os filtros de estado resolvem esse problema rastreando todas as conexões TCP de entrada em uma tabela de conexão. Isso é possível porque o *firewall* pode notar o início de uma nova conexão observando uma apresentação de três vias (SYN, SYNACK e ACK); ele pode observar o fim de uma conexão ao ver um pacote FIN para a conexão. O *firewall* também consegue (de forma conservadora) admitir que a conexão está finalizada quando não observou nenhuma atividade no decorrer da conexão, digamos, por 60 segundos. Um exemplo de tabela de conexão para um *firewall* é mostrado na Tabela 8.7. Essa tabela indica que, no momento, há três conexões TCP em andamento, as quais foram iniciadas dentro da organização. Ademais, o filtro de estado inclui uma nova coluna, "verificar conexão", em sua lista de controle de acesso, como mostrado na Tabela 8.8. Observe que essa tabela é idêntica à lista de controle de acesso da Tabela 8.6, exceto por ela indicar que a conexão deve ser verificada para duas das regras.

Vamos analisar alguns exemplos e ver como a tabela de conexão e a lista de controle de acesso funcionam em conjunto. Suponha que um atacante tente enviar um pacote defeituoso para a rede da organização por meio de um datagrama com porta de origem TCP 80 e com o *flag* ACK marcado. Suponha ainda que ele possua um número de porta de origem 12543 e endereço IP remetente 150.23.23.155. Quando o pacote chega ao *firewall*, este verifica a lista de controle de acesso da Tabela 8.8, que indica que a tabela de conexão deve também ser verificada antes de permitir que esse pacote entre na rede da organização. O *firewall* verifica devidamente a tabela de conexão e observa que esse pacote não faz parte de uma conexão TCP em andamento, e o rejeita. Como segundo exemplo, imagine que um usuário interno queira navegar em um site externo. Como o usuário primeiro envia um segmento TCP SYN, sua conexão TCP é registrada na tabela de conexão. Quando o servidor envia pacotes de volta (com o bit ACK necessariamente definido), o *firewall* verifica a tabela e observa que uma conexão correspondente está em andamento. O *firewall*, então, deixará esses pacotes passarem, sem interferir na navegação do usuário interno.

TABELA 8.7 TABELA DE CONEXÃO PARA O FILTRO DE ESTADO

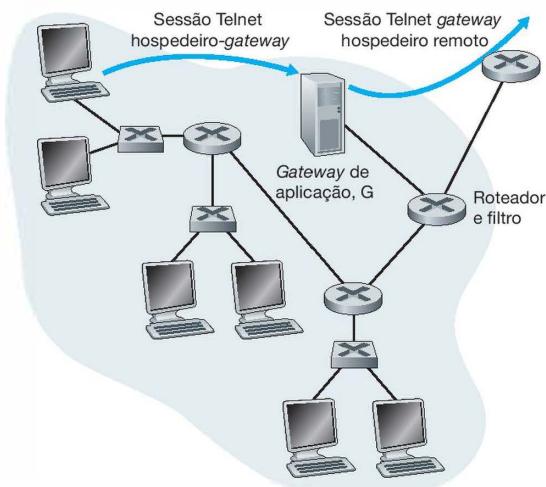
Endereço de origem	Endereço de destino	Porta de origem	Porta de destino
222.22.1.7	37.96.87.123	12699	80
222.22.93.2	199.1.205.23	37654	80
222.22.65.143	203.77.240.43	48712	80

TABELA 8.8 LISTA DE CONTROLE DE ACESSO PARA FILTRO DE ESTADO

Ação	Endereço de origem	Endereço de destino	Protocolo	Porta de origem	Porta de destino	Bit de flag	Verificar conexão
Permitir	222.22/16	Fora de 222.22/16	TCP	>1023	80	Qualquer um	
Permitir	Fora de 222.22/16	222.22/16	TCP	80	>1023	ACK	X
Permitir	222.22/16	Fora de 222.22/16	UDP	>1023	53	—	
Permitir	Fora de 222.22/16	222.22/16	UDP	53	>1023	—	X
Negar	Todos	Todos	Todos	Todos	Todos	Todos	

Gateway de aplicação

Nos exemplos que acabamos de mostrar, vimos que a filtragem de pacotes permite que uma organização faça uma filtragem grosseira de conteúdos de cabeçalhos IP e TCP/UDP, incluindo endereços IP, números de porta e bits de reconhecimento. Mas, e se a organização quiser fornecer o serviço Telnet a um conjunto restrito de usuários internos (em vez de a endereços IP)? E se quiser que esses usuários privilegiados se autentiquem antes de obter permissão para criar sessões Telnet com o mundo externo? Essas tarefas estão além das capacidades de

um filtro. Na verdade, informações sobre a identidade de usuários internos não estão incluídas nos cabeçalhos IP/TCP/UDP; elas estão nos dados da camada de aplicação.

Para assegurar um nível mais refinado de segurança, os *firewalls* têm de combinar filtro de pacotes com *gateways* de aplicação. *Gateways* de aplicação fazem mais do que examinar cabeçalhos IP/TCP/UDP e tomam decisões com base em dados da aplicação. Um ***gateway* de aplicação** é um servidor específico de aplicação, através do qual todos os dados da aplicação (que entram e que saem) devem passar. Vários *gateways* de aplicação podem executar no mesmo hospedeiro, mas cada *gateway* é um servidor separado, com seus próprios processos.

Para termos uma ideia melhor desses *gateways*, vamos projetar um *firewall* que permite a apenas um conjunto restrito de usuários executar Telnet para o exterior e impede que todos os clientes externos executem Telnet para o interior. Essa política pode ser aplicada pela execução da combinação de um filtro de pacotes (em um roteador) com um *gateway* de aplicação de Telnet, como mostra a Figura 8.34. O filtro do roteador está configurado para bloquear todas as conexões Telnet, exceto as que se originam do endereço IP do *gateway* de aplicação. Essa configuração de filtro força todas as conexões Telnet de saída a passarem pelo *gateway* de aplicação. Considere agora um usuário interno que quer executar Telnet com o mundo exterior. Primeiro, ele tem de estabelecer uma sessão Telnet com o *gateway* de aplicação. Uma aplicação que está executando no *gateway* — e que fica à escuta de sessões Telnet que entram — solicita ao usuário sua identificação e senha. Quando o usuário fornece essas informações, o *gateway* de aplicação verifica se ele tem permissão para executar Telnet com o mundo exterior. Se não tiver, a conexão Telnet do usuário interno ao *gateway* será encerrada pelo *gateway*. Se o usuário tiver permissão, o *gateway* (1) pedirá ao usuário o nome do computador externo com o qual ele quer se conectar, (2) estabelecerá uma sessão Telnet entre o *gateway* e o hospedeiro externo e (3) repassará ao hospedeiro externo todos os dados que chegam do usuário e ao usuário todos os dados que chegam do hospedeiro externo. Assim, o *gateway* de aplicação Telnet não só autoriza o usuário, mas também atua como um servidor Telnet e um cliente Telnet, repassando informações entre o usuário e o servidor Telnet remoto. Note que o filtro permitirá a etapa 2, porque é o *gateway* que inicia a conexão Telnet com o mundo exterior.

Redes internas frequentemente têm vários *gateways* de aplicação, como *gateways* para Telnet, HTTP, FTP e e-mail. De fato, o servidor de correio (veja a Seção 2.4) e o *cache* Web de uma organização são *gateways* de aplicação.

Gateways de aplicação não estão isentos de desvantagens. Primeiro, é preciso um *gateway* de aplicação diferente para cada aplicação. Segundo, há um preço a pagar em termos de desempenho, visto que todos os dados serão

FIGURA 8.34 ***FIREWALL* COMPOSTO DE UM *GATEWAY* DE APLICAÇÃO E UM FILTRO**

HISTÓRIA

Anonimato e privacidade

Suponha que você queira visitar aquela polêmica página Web (por exemplo, o site de um ativista político) e você (1) não quer revelar seu endereço IP à página Web, (2) não quer que o seu ISP (que pode ser o de sua casa ou do escritório) saiba que você está visitando esse site, e (3) não quer que o seu IP local veja os dados que você está compartilhando com o site. Se você usar o método tradicional de conexão direta à página Web sem nenhuma criptografia, então falhará em seus três objetivos. Mesmo que use SSL, você falhará nas duas primeiras questões: seu endereço IP de origem é apresentado à página Web em todo datagrama enviado; e o endereço de destino de cada pacote enviado pode facilmente ser analisado pelo seu ISP local.

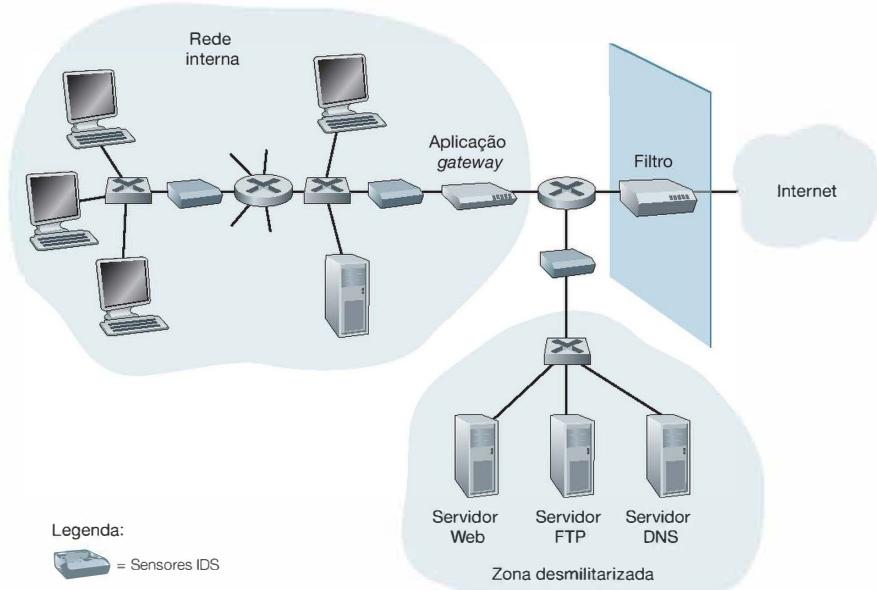
Para obter anonimato e privacidade, você pode usar uma combinação de um servidor *proxy* confiável e SSL, como mostrado na Figura 8.35. Com essa técnica, você faz uma conexão SSL com o *proxy* confiável. Depois envia, nessa conexão SSL, uma solicitação HTTP para o site desejado. Quando o *proxy* receber essa solicitação HTTP criptografada por SSL, ele a decodificará e encaminhará o texto claro da solicitação HTTP à página Web. Esta responde ao *proxy*, que por sua vez encaminha a resposta a você pelo SSL. Como a página Web só vê o endereço IP do *proxy*, e não o do seu cliente, você está de fato obtendo um acesso anônimo à página Web. E devido ao tráfego entre você e o *proxy* ser criptografado, seu ISP local não pode invadir sua privacidade ao logar no site que você visitou ou gravar os dados que estavam sendo compartilhados. Hoje, muitas empresas disponibilizam tais serviços *proxy* (como a proxify.com).

É claro que, ao usar esta solução, seu *proxy* saberá tudo: seu endereço IP e o endereço IP do site que você está visitando; pode ver todo o tráfego em texto claro compartilhado entre você e a página. Uma técnica melhor, adotada pelo serviço de anonimato e privacidade TOR, é sequenciar seu tráfego através de uma série de servidores *proxys* que não compartilham informações entre si [TOR 2012]. Em particular, o TOR permite que indivíduos independentes contribuam com *proxys* para seu acervo. Quando um usuário se conecta a um servidor usando o TOR, ele escolhe aleatoriamente (de seu acervo de *proxys*) uma corrente de três *proxys* e sequencia todo o tráfego entre cliente e servidor por essa corrente. Dessa maneira, supondo que os *proxys* não trocam informações entre si, ninguém percebe que ocorreu uma comunicação entre seu endereço IP e a página da Web desejada. Além disso, apesar de o texto claro ser enviado entre o último *proxy* e o servidor, o último *proxy* não sabe qual endereço IP está enviando ou recebendo o texto claro.

FIGURA 8.35 FORNECENDO ANONIMATO E PRIVACIDADE COM UM *PROXY*

repassados por meio do *gateway*. Isso se torna uma preocupação em particular quando vários usuários ou aplicações estão utilizando o mesmo *gateway*. Por fim, o software cliente deve saber como entrar em contato com o *gateway* quando o usuário fizer uma solicitação, e deve saber como dizer ao *gateway* de aplicação a qual servidor se conectar.

8.9.2 Sistemas de detecção de invasão

Acabamos de ver que um filtro de pacotes (tradicional e de estado) inspeciona campos de cabeçalho IP, TCP, UDP e ICMP quando está decidindo quais pacotes deixará passar através do *firewall*. No entanto, para detectar muitos tipos de ataque, precisamos executar uma **inspeção profunda de pacote**, ou seja, precisamos olhar através dos campos de cabeçalho e dentro dos dados da aplicação que o pacote carrega. Como vimos na Seção 8.9.1, *gateways* de aplicação frequentemente fazem inspeções profundas de pacote. Mas um *gateway* de aplicação só executa isso para uma aplicação específica.

Decerto existe espaço para mais um dispositivo — um dispositivo que não só examina os cabeçalhos de todos os pacotes ao passar por eles (como um filtro de pacotes), mas também executa uma inspeção profunda de pacote (diferente do filtro de pacotes). Quando tal dispositivo observa o pacote suspeito, ou uma série de pacotes suspeitos, ele impede que tais pacotes entrem na rede organizacional. Ou, quando a atividade só é vista como suspeita, o dispositivo pode deixar os pacotes passarem, mas envia um alerta ao administrador de rede, que pode examinar o tráfego minuciosamente e tomar as ações necessárias. Um dispositivo que gera alertas quando observa tráfegos potencialmente mal-intencionados é chamado de **sistema de detecção de invasão** (**IDS**, do inglês *intrusion detection system*). Um dispositivo que filtra o tráfego suspeito é chamado de **sistema de prevenção de invasão** (**IPS**, do inglês *intrusion prevention system*). Nesta seção, estudaremos ambos os sistemas — IDS e IPS —, já que o mais interessante aspecto técnico desses sistemas é como eles detectam tráfego suspeito (em vez de enviarem alertas ou abandonarem pacotes). Daqui para a frente vamos nos referir ao sistema IDS e ao sistema IPS como sistema IDS.

Um IDS pode ser usado para detectar uma série de tipos de ataques, incluindo mapeamento de rede (provindo, por exemplo, de nmap), varreduras de porta, varreduras de pilha TCP, ataques de DoS, ataques de inundação de largura de banda, worms e vírus, ataques de vulnerabilidade de OS e ataques de vulnerabilidade de aplicações. (Veja, na Seção 1.6, um tutorial sobre ataques de rede.) Hoje, milhares de organizações empregam sistemas de IDS. Muitos desses sistemas são patenteados, comercializados pela Cisco, Check Point, e outros fornecedores de equipamentos de segurança. Mas muitos dos sistemas de IDS implementados são de domínio público, como o extremamente popular Snort IDS (o qual discutiremos em breve).

Uma organização pode pôr em prática um ou mais sensores IDS em sua rede organizacional. A Figura 8.36 mostra uma organização que tem três sensores IDS. Quando múltiplos sistemas são executados, eles costumam trabalhar em harmonia, enviando informações sobre atividades de tráfegos suspeitos ao processador IDS central, que as coleta e integra e envia alarmes aos administradores da rede quando acharem apropriado. Na Figura 8.36, a organização dividiu sua rede em duas regiões: uma de segurança máxima, protegida por um filtro de pacotes e um *gateway* de aplicação e monitorada por sensores IDS; e uma região de segurança baixa — referida como **zona desmilitarizada** (**DMZ**, do inglês *demilitarized zone*) — protegida apenas por um filtro de pacotes, mas também monitorada por sensores IDS. Observe que a DMZ inclui os servidores da organização que precisam se comunicar com o mundo externo, como um servidor Web e seus servidores autoritativos.

Neste ponto, você deve estar imaginando: mas por que sensores IDS? Por que não colocar um sensor IDS logo atrás do filtro de pacotes (ou até integrá-lo ao filtro de pacotes) da Figura 8.36? Logo veremos que um IDS não só precisa fazer uma inspeção profunda do pacote, como também comparar cada pacote que passa com milhares de "assinaturas"; isso pode ser um volume de processamento significativo, em particular se a organização recebe gigabits/s de tráfego da Internet. Ao colocar sensores IDS mais à frente, cada sensor só vê uma fração do tráfego da organização, e pode facilmente acompanhar o ritmo. No entanto, hoje existem sistemas IDS e IPS de alto desempenho, e muitas organizações podem acompanhar com apenas um sensor localizado próximo ao roteador de acesso.

Sistemas IDS são classificados de modo geral tanto como **sistemas baseados em assinatura**, ou **sistemas baseados em anomalia**. Um IDS baseado em assinatura mantém um banco de dados extenso de ataques de assi-

FIGURA 8.36 UMA ORGANIZAÇÃO IMPLEMENTANDO UM FILTRO, UMA APLICAÇÃO *GATEWAY* E SENSORES IDS

naturas. Cada assinatura é um conjunto de regras relacionadas a uma atividade de invasão. Uma assinatura pode ser uma lista de características sobre um único pacote (por exemplo, números de portas de origem e destino, tipo de protocolo, e uma sequência de bits em uma carga útil de um pacote), ou estar relacionada a uma série de pacotes. As assinaturas são normalmente criadas por engenheiros habilidosos em segurança de rede que tenham pesquisado ataques conhecidos. O administrador de rede de uma organização pode personalizar as assinaturas ou inserir suas próprias no banco de dados.

Operacionalmente, uma IDS baseada em assinatura analisa cada pacote que passa, comparando cada um com as assinaturas no banco de dados. Se um pacote (ou uma série deles) corresponder a uma assinatura no banco de dados, o IDS gera um alerta. O alerta pode ser enviado ao administrador da rede por uma mensagem de correio eletrônico, pode ser enviado ao sistema de gerenciamento da rede, ou pode simplesmente ser registrado para futuras inspeções.

Apesar de os sistemas IDS baseados em assinaturas serem amplamente executados, eles têm uma série de limitações. Acima de tudo, eles requerem conhecimento prévio do ataque para gerar uma assinatura precisa. Ou seja, um IDS baseado em assinatura é completamente cego a novos ataques que ainda não foram registrados. Outra desvantagem é que, mesmo que uma assinatura combine, isso pode não ser o resultado de um ataque, mas mesmo assim um alarme é gerado. Por fim, pelo fato de cada pacote ser comparado com uma extensa coleção de assinaturas, o IDS fica atarefado com o processamento e deixa de detectar muitos pacotes malignos.

Um IDS baseado em anomalias cria um perfil de tráfego enquanto observa o tráfego em operação normal. Ele procura então por fluxos de pacotes que são estatisticamente incomuns, por exemplo, uma porcentagem irregular de pacotes ICMP ou um crescimento exponencial de análises de porta e varreduras de *ping*. O mais interessante sobre sistemas de IDS baseados em anomalias é que eles não recorrem a conhecimentos prévios de outros ataques — ou seja, potencialmente, eles conseguem detectar novos ataques, que não foram documentados. Por outro lado, é um problema extremamente desafiador distinguir o tráfego normal de tráfegos estatisticamente incomuns. Até hoje, a maioria das implementações de IDS são principalmente baseadas em assinaturas, apesar de algumas terem alguns recursos baseados em anomalias.

Snort

Snort é um IDS de código aberto, de domínio público, com centenas de milhares de execuções [Snort, 2012; Koziol, 2003]. Ele pode ser executado em plataformas Linux, UNIX e Windows. Usa uma interface libpcap de análise genérica, que também é empregada pelo Wireshark e muitas outras ferramentas de análises de pacotes. Pode facilmente lidar com 100 Mbits/s de tráfego; para instalações com velocidades de tráfego de gigabit/s, múltiplos sensores Snort serão necessários.

Para termos uma ideia melhor do Snort, vamos observar o exemplo de uma assinatura Snort:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any
(msg:"ICMP PING NMAP"; dsize: 0; itype: 8;)
```

Esta assinatura é compatível com qualquer pacote ICMP que entre na rede da organização (`$HOME_NET`) e que venha do exterior (`$EXTERNAL_NET`), seja do tipo 8 (ping ICMP), e tenha uma carga útil vazia (dsize=0). Já que o nmap (veja a Seção 1.6) gera pacotes *ping* com características específicas, essa assinatura é projetada para detectar varreduras de *ping* usadas pelo nmap. Quando um pacote corresponde a essa assinatura, o Snort gera um alerta que inclui a mensagem "`ICMP PING NMAP`".

Talvez o mais impressionante sobre o Snort seja a vasta comunidade de usuários e especialistas em segurança que mantêm sua base de dados de assinaturas. Normalmente, em algumas horas do novo ataque, a comunidade escreve e lança uma assinatura de ataque, que então é transferida via download pelas centenas de milhares de execuções de Snort ao redor do mundo. Além disso, ao usarem a sintaxe da assinatura do Snort, os administradores de rede podem criar suas próprias assinaturas a fim de satisfazer as necessidades da organização, modificando assinaturas existentes ou criando novas.

8.10 RESUMO

Neste capítulo, examinamos os diversos mecanismos que Bob e Alice, os amantes secretos, podem usar para se comunicar com segurança. Vimos que estão interessados em confidencialidade (para que somente eles possam entender o conteúdo de uma mensagem transmitida), autenticação do ponto final (para terem a certeza de que estão falando um com o outro) e integridade de mensagem (para terem certeza de que suas mensagens não sejam alteradas em trânsito). É claro que a necessidade de comunicação segura não está limitada a amantes secretos. Na verdade, vimos nas seções 8.5 a 8.8 que a segurança é necessária em várias camadas de uma arquitetura de rede para proteção contra bandidos que têm à mão um grande arsenal de ataques possíveis.

Na primeira parte deste capítulo, apresentamos vários princípios subjacentes à comunicação segura. Na Seção 8.2, examinamos as técnicas para criptografar e decriptografar dados, incluindo criptografia de chaves simétricas e criptografia de chaves públicas. O DES e o RSA foram examinados como estudos de caso específicos dessas duas classes mais importantes de técnicas de criptografia em uso nas redes de hoje.

Na Seção 8.3, examinamos duas técnicas para fornecer a integridade da mensagem: códigos de autenticação de mensagem (MACs) e assinaturas digitais. As duas têm uma série de paralelos. Ambas usam funções *hash* criptografadas e ambas permitem que verifiquemos a origem, assim como a integridade da própria mensagem. Uma diferença importante é que os MACs não recorrem à criptografia, ao passo que as assinaturas digitais necessitam de uma infraestrutura de chave pública. Ambas as técnicas são muito usadas na prática, como vimos nas seções 8.5 a 8.8. Além disso, as assinaturas digitais são usadas para criar certificados digitais, os quais são importantes para a verificação da validade de uma chave pública. Na Seção 8.4, vimos também a autenticação do ponto final e como *nonces* podem ser usados para impedir ataques de repetição.

Nas seções 8.5 a 8.8, examinamos diversos protocolos de segurança de rede que são usados extensivamente na prática. Vimos que uma criptografia de chave simétrica está no núcleo do PGP, SSL, IPsec e segurança sem fio. Vimos que uma criptografia pública é crucial para ambos PGP e SSL. Estudamos que o PGP usa assinaturas digitais para a integridade da mensagem, ao passo que SSL e IPsec usam MACs. Agora que entendemos os princípios

básicos da criptografia, e tendo estudado como esses princípios são usados, você deve estar pronto para projetar seus próprios protocolos de segurança de rede!

Munidos das técnicas abordadas nas seções 8.2 a 8.8, Bob e Alice podem se comunicar com segurança (esperamos que eles sejam estudantes de rede que aprenderam o que este livro ensinou e consigam, dessa maneira, evitar que seus encontros secretos sejam descobertos por Trudy!). Porém, a confiabilidade é apenas uma pequena parte do quadro da segurança na rede. Como vimos na Seção 8.9, o foco está se concentrando cada vez mais em garantir a segurança da infraestrutura da rede contra o ataque potencial dos bandidos. Assim, na última parte deste capítulo, estudamos *firewalls* e sistemas IDS que inspecionam pacotes que entram e saem da rede de uma organização.

Este capítulo cobriu diversos fundamentos, enquanto focava nos tópicos mais importantes sobre a segurança nas redes modernas. Os leitores que desejam mais informações podem investigar as referências citadas. Em particular, recomendamos Skoudis [2006] sobre ataques e segurança operacional, Kaufmann [1995] para criptografia e como ela se aplica à segurança de redes, Rescorla [2001] para uma explicação profunda, mas de fácil compreensão sobre SSL e Edney [2003] para uma discussão completa sobre segurança 802.11, incluindo uma investigação sobre WEP e suas falhas.

EXERCÍCIOS DE FIXAÇÃO E PERGUNTAS

Questões de revisão do Capítulo 8

SEÇÃO 8.1

R1. Quais são as diferenças entre confidencialidade de mensagem e integridade de mensagem? É possível ter confidencialidade sem integridade? É possível ter integridade sem confidencialidade? Justifique sua resposta.

R2. Equipamentos da Internet (roteadores, comutadores, servidores DNS, servidores Web, sistemas do usuário final etc.) frequentemente precisam se comunicar com segurança. Dê três exemplos específicos de pares de equipamentos da Internet que precisem de uma comunicação segura.

SEÇÃO 8.2

R3. Da perspectiva de um serviço, qual é uma diferença importante entre um sistema de chave simétrica e um sistema de chave pública?

R4. Suponha que um intruso tenha uma mensagem criptografada, bem como a versão decodificada dessa mensagem. Ele pode montar um ataque somente com texto cifrado, um ataque com texto aberto conhecido ou um ataque com texto aberto escolhido?

R5. Considere uma cifra de 8 blocos. Quantos blocos de entrada possíveis uma cifra tem? Quantos mapeamentos possíveis existiriam? Se analisarmos cada mapeamento como uma chave, então quantas chaves essa cifra teria?

R6. Suponha que N pessoas queiram se comunicar com cada uma das outras $N — 1$ pessoas usando criptografia de chaves simétricas. Todas as comunicações entre quaisquer duas pessoas, i e j, são visíveis para todas as outras do grupo de N, e nenhuma outra pessoa desse grupo pode decodificar suas comunicações. O sistema, como um todo, requer quantas chaves? Agora, suponha que seja usada criptografia de chaves públicas. Quantas chaves serão necessárias nesse caso?

R7. Suponha que n = 10.000, a = 10.023 e b = 10.004. Use uma identidade da aritmética modular para calcular $(a \cdot b)$ mod n de cabeça.

R8. Suponha que você queira criptografar a mensagem 10101111 criptografando um número decimal que corresponda a essa mensagem. Qual seria esse número decimal?

SEÇÕES 8.3–8.4

R9. De que maneira um *hash* fornece um melhor controle de integridade da mensagem do que uma soma de verificação (como a soma de verificação da Internet)?

R10. Você pode decodificar o *hash* de uma mensagem a fim de obter a mensagem original? Explique sua resposta.

R11. Considere a variação de um algoritmo MAC (Figura 8.9), em que o transmissor envia $(m, H(m) + s)$, sendo $H(m) + s$ a concatenação de $H(m)$ e s. Essa variação é falha? Por que ou por que não?

R12. O que significa afirmar que um documento é verificável e não falsificável?

R13. De que modo um resumo de mensagem criptografado por chave pública proporciona uma assinatura digital melhor do que utilizar a mensagem criptografada com chave pública?

R14. Suponha que a certificador.com crie um certificado para a alguem.com. Normalmente, o certificado inteiro seria criptografado com a chave pública de certificador.com. Verdadeiro ou falso?

R15. Suponha que Alice tenha uma mensagem pronta para enviar para qualquer pessoa que pedir. Milhares de pessoas querem ter a mensagem da Alice, mas cada uma quer ter certeza da integridade da mensagem. Nesse contexto, você acha que é mais apropriado um esquema baseado em MAC ou um baseado em assinatura digital? Por quê?

R16. Qual é a finalidade de um *nonce* em um protocolo de identificação de ponto final?

R17. O que significa dizer que um *nonce* é um valor usado uma vez por toda a vida? Pelo tempo de vida de quem?

R18. O esquema de integridade de mensagem baseado no HMAC é suscetível a ataques de repetição? Se for, como um *nonce* pode ser incorporado ao esquema para remover essa suscetibilidade?

SEÇÕES 8.5–8.8

R19. Suponha que Bob receba uma mensagem PGP de Alice. Como o Bob sabe com certeza que Alice criou a mensagem (e não Trudy, por exemplo)? O PGP usa um MAC para integridade da mensagem?

R20. Em um registro SSL, há um campo para uma sequência de números SSL. Verdadeiro ou falso?

R21. Qual é a finalidade de um *nonce* em um protocolo de autenticação em uma apresentação SSL?

R22. Suponha que uma sessão SSL utilize uma cifra de bloco com CBC. Verdadeiro ou falso: o servidor envia o IV ao cliente de forma aberta.

R23. Suponha que Bob inicie uma conexão TCP com Trudy, que está fingindo ser Alice. Durante a apresentação, Trudy envia a Bob um certificado de Alice. Em qual etapa do algoritmo de apresentação SSL Bob descobrirá que não está se comunicando com Alice?

R24. Considere uma cadeia de pacotes do Hospedeiro A ao Hospedeiro B usando IPsec. Em geral, um novo SA será estabelecido para cada pacote enviado na cadeia. Verdadeiro ou falso?

R25. Suponha que o TCP esteja sendo executado por IPsec entre a matriz e a filial na Figura 8.28. Se o TCP retransmitir o mesmo pacote, então os dois pacotes correspondentes enviados por pacotes R1 terão o mesmo número sequencial no cabeçalho ESP. Verdadeiro ou falso?

R26. Um SA IKE e um SA IPsec são a mesma coisa. Verdadeiro ou falso?

R27. Considere um WEP para 802.11. Suponha que a informação seja 10101100 e o fluxo de chaves seja 1111000. Qual é o texto cifrado resultante?

R28. Em WEP, um IV é enviado em aberto em cada quadro. Verdadeiro ou falso?

SEÇÃO 8.9

R29. Um filtro de pacotes com estado mantém duas estruturas de dados. Nomeie-as e descreva resumidamente o que elas fazem.

R30. Considere um filtro de pacotes tradicional (sem estado). Ele pode filtrar pacotes baseado em bits de *flag* TCP assim como em outros campos de cabeçalho. Verdadeiro ou falso?

R31. Em um filtro de pacotes tradicional, cada interface pode ter sua própria lista de controle de acesso. Verdadeiro ou falso?

R32. Por que uma aplicação de *gateway* deve trabalhar junto com o filtro de roteador para ser eficaz?

R33. IDSs baseados em assinaturas e IPSs inspecionam a carga útil de segmentos TCP e UDP. Verdadeiro ou falso?

PROBLEMAS

P1. Usando a cifra monoalfabética da Figura 8.3, codifique a mensagem "This is an easy problem" (este é um problema fácil). Decodifique a mensagem "rmij'u uamu xyj".

P2. Mostre que o ataque com texto aberto conhecido de Trudy em que ela conhece os pares de tradução (texto cifrado, texto aberto) para sete letras reduz em aproximadamente 10^9 o número de possíveis substituições a verificar no exemplo apresentado na Seção 8.2.1.

P3. Considere o sistema polialfabético mostrado na Figura 8.4. Um ataque com texto aberto escolhido que consiga obter a codificação da mensagem "The quick fox jumps over the lazy brown dog" é suficiente para decifrar todas as mensagens? Explique sua resposta.

P4. Considere o bloco cifrado na Figura 8.5. Suponha que cada bloco cifrado T_i simplesmente inverta a ordem dos oito bits de entrada (então, por exemplo, 11110000 se torna 00001111). Além disso, imagine que o misturador de 64 bits não modifique qualquer bit (de modo que o valor de saída de *m*-ésimo bit seja igual ao valor de entrada do *m*-ésimo bit). (a) Sendo $n = 3$ e a entrada original de 64 bits igual a 10100000 repetidos oito vezes, qual é o valor da saída? (b) Repita a parte (a), mas agora troque o último bit da entrada original de 64 bits de 0 para 1. (c) Repita as partes (a) e (b), mas agora suponha que o misturador de 64 bits inverta a ordem de 64 bits.

P5. Considere o bloco cifrado da Figura 8.5 Para uma determinada "chave", Alice e Bob precisariam ter 8 tabelas, cada uma com 8 bits por 8 bits. Para Alice (ou Bob) armazenar todas as oito tabelas, quantos bits de armazenamento são necessários? Como esse número se compara com o número de bits necessários para um bloco cifrado de tabela cheia de 64 bits?

P6. Considere o bloco cifrado de 3 bits da Tabela 8.1. Suponha que o texto aberto seja 100100100. (a) Inicialmente suponha que o CBC não seja usado. Qual é o texto cifrado resultante? (b) Imagine que Trudy analise o texto cifrado. Supondo que ela saiba que um bloco cifrado de bits está sendo usado sem o CBC (mas ela não sabe a cifra específica), o que ela pode suspeitar? (c) Agora suponha que o CBC é usado com IV = 111. Qual é o texto cifrado resultante?

P7. (a) Usando RSA, escolha $p = 3$ e $q = 11$ e codifique a palavra "dog", criptografando cada letra em separado. Aplique o algoritmo de decriptação à versão criptografada para recuperar a mensagem original em texto aberto. (b) Repita a parte (a), mas agora criptografe "dog" como uma mensagem m.

P8. Considere o RSA com $p = 5$ e $q = 11$.

a. Quais são n e z?

b. Seja e igual a 3. Por que esta é uma escolha aceitável para e?

c. Encontre d tal que $de = 1 \pmod z$ e $d < 160$.

d. Criptografe a mensagem $m = 8$ usando a chave (n, e). Seja c o texto cifrado correspondente. Mostre todo o processo. *Dica:* Para simplificar os cálculos, use este fato:

$$[(a \bmod n) \bullet (b \bmod n)] \bmod n = (a \bullet b) \bmod n$$

P9. Neste problema, exploraremos o algoritmo criptografado Diffie-Hellman (DH) de chave pública, o qual permite que duas entidades concordem em uma chave compartilhada. O algoritmo DH faz uso de um número primo grande p e outro número grande g, menor que p. Ambos, p e g, são públicos (de modo que um atacante os conheça). Em DH, Alice e Bob escolhem, independentemente, chaves secretas, S_A e S_B. Alice então calcula sua chave pública, T_A, elevando g a S_A e tomando o mod p. Bob, similarmente, calcula sua própria chave pública, T_B, elevando g a S_B e tomando o módulo p. Então, Alice e Bob trocam suas chaves públicas pela

Internet. Alice calcula a chave secreta compartilhada S ao elevar T_B a S_A e tomando o módulo p. Bob, de modo semelhante, calcula a chave compartilhada S' ao elevar T_A a S_B e tomando o módulo p.

a. Prove que, no geral, Alice e Bob obtêm a mesma chave simétrica, ou seja, prove que $S = S'$.

b. Com $p = 11$ e $g = 2$, suponha que Alice e Bob escolham chaves privadas $S_A = 5$ e $S_B = 12$, respectivamente. Calcule as chaves públicas de Alice e Bob, T_A e T_B. Mostre todo o processo.

c. Seguindo a parte (b), calcule S como a chave simétrica compartilhada. Mostre todo o processo.

d. Forneça um diagrama de tempo que mostre como o Diffie-Hellman pode ser atacado por um homem do meio. O diagrama de tempo deve ter três linhas verticais, uma para Alice, uma para Bob e uma para a atacante Trudy.

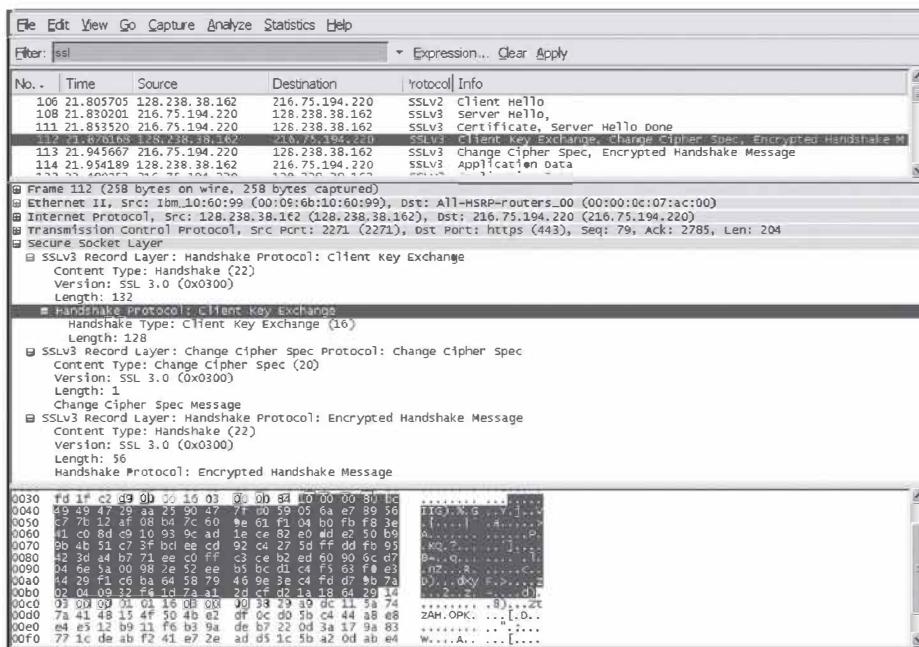
P10. Suponha que Alice queira se comunicar com o Bob usando uma chave simétrica criptografada usando uma chave de sessão K_S. Na Seção 8.2, aprendemos que uma chave pública criptografada pode ser usada para distribuir a chave de sessão da Alice para Bob. Neste problema, exploramos como uma chave de sessão pode ser distribuída — sem uma chave pública criptografada — usando um centro de distribuição de chaves (KDC, do inglês *key distribution center*). O KDC é um servidor que compartilha uma única chave simétrica secreta com cada usuário registrado. Para Alice e Bob, indique essas chaves como $K_{A\text{-}KDC}$ e $K_{B\text{-}KDC}$. Projete um esquema que use o KDC para distribuir K_S a Alice e Bob. Seu esquema deverá usar três mensagens para distribuir uma chave de sessão: a de Alice ao KDC; a do KDC a Alice; e por fim a de Alice para Bob. A primeira mensagem é $K_{A\text{-}KDC}(A, B)$. Usando a notação, $K_{A\text{-}KDC}$, $K_{B\text{-}KDC}$, S, A e B, responda às seguintes perguntas.

a. Qual é a segunda mensagem?

b. Qual é a terceira mensagem?

P11. Calcule uma terceira mensagem, diferente das duas mensagens na Figura 8.8, que tenha a mesma soma de verificação dessas duas mensagens na figura citada.

P12. Suponha que Alice e Bob compartilhem duas chaves secretas: uma chave de autenticação S_1 e uma chave criptografada simétrica S_2. Aumente a Figura 8.9 para que ambas, integridade e confidencialidade, sejam fornecidas.

P13. No protocolo de distribuição de arquivo P2P BitTorrent (veja o Capítulo 2), a semente quebra o arquivo em blocos, e os pares redistribuem os blocos uns aos outros. Sem proteção alguma, um atacante pode facilmente causar destruição em um *torrent* ao se mascarar como se fosse um par benevolente e enviar blocos falsos aos outros conjuntos de pares no *torrent*. Esse pares que não são suspeitos redistribuem os blocos falsos a mais outros pares, que distribuem os blocos falsos aos outros pares. Sendo assim, é importante para o BitTorrent ter um mecanismo que permita aos pares verificar a integridade de um bloco, para que não distribuam blocos falsos. Suponha que, quando um par se junta a um *torrent*, de início pega um arquivo .torrent de uma fonte *inteiramente* confiável. Descreva um esquema simples que permita que os pares verifiquem a integridade dos blocos.

P14. O protocolo de roteamento OSPF usa um MAC em vez de assinaturas digitais para fornecer a integridade da mensagem. Por que você acha que foi escolhido o MAC em vez de assinaturas digitais?

P15. Considere nosso protocolo de autenticação da Figura 8.18, no qual Alice se autentica para Bob, e que vimos que funciona bem (isto é, não encontramos nenhuma falha nele). Agora suponha que, enquanto Alice está se autenticando para Bob, este deve se autenticar para Alice. Apresente um cenário no qual Trudy, fazendo-se passar por Alice, agora pode se autenticar para Bob como se fosse Alice. (*Dica*: considere que a sequência de operações do protocolo, uma com Trudy e outra com Bob iniciando a sequência, pode ser intercalada arbitrariamente. Preste atenção particular ao fato de que Bob e Alice usarão um *nonce* e que, caso não se tome cuidado, o mesmo *nonce* pode ser utilizado de forma maliciosa.)

P16. Uma questão natural é se podemos usar um *nonce* e a criptografia de chave pública para resolver o problema de autenticação do ponto final na Seção 8.4. Considere o seguinte protocolo natural: (1) Alice envia a mensagem "`Eu sou Alice`" a Bob. (2) Bob escolhe um *nonce*, R, e o envia a Alice. (3) Alice usa sua chave *privada* para criptografar o *nonce* e envia o valor resultante a Bob. (4) Bob aplica a chave pública de Alice à mensagem recebida. Assim, Bob calcula R e autentica Alice.

a. Faça um diagrama desse protocolo, usando a notação para chaves públicas e privadas, empregada neste livro.

b. Suponha que os certificados não sejam usados. Descreva como Trudy pode se tornar uma "mulher no meio", interceptando as mensagens de Alice e depois se passando por Alice para Bob.

P17. A Figura 8.19 mostra as operações que Alice deve realizar com PGP para fornecer confidencialidade, autenticação e integridade. Faça um diagrama das operações correspondentes que Bob precisa executar no pacote recebido por Alice.

P18. Suponha que Alice queira enviar um e-mail a Bob. Bob tem um par de chave pública-privada (K_B^+,K_B^-), e Alice tem o certificado de Bob. Mas Alice não tem um par de chave pública, privada. Alice e Bob (e o mundo inteiro) compartilham a mesma função de *hash* H(.).

a. Nessa situação, é possível projetar um esquema para que Bob possa verificar que Alice criou a mensagem? Se sim, mostre como, com um diagrama de bloco para Alice e Bob.

b. É possível projetar um esquema que forneça confidencialidade para enviar a mensagem de Alice a Bob? Se sim, mostre como com um diagrama de bloco para Alice e Bob.

P19. Considere a saída Wireshark a seguir para uma parte de uma sessão SSL.

a. O pacote Wireshark 112 é enviado pelo cliente ou pelo servidor?

b. Qual o número IP do servidor e seu número de porta?

c. Considerando que nenhuma perda ou retransmissão ocorreram, qual será a sequência de números do próximo segmento TCP enviado pelo cliente?

d. Quantos registros SSL existem no pacote Wireshark 112?

e. O pacote 112 contém um Segredo Mestre ou um Segredo Mestre Criptografado, ou nenhum?

f. Supondo que o campo do tipo na apresentação é de 1 byte e cada campo de comprimento é de 3 bytes, quais são os valores do primeiro e do último bytes do Segredo Mestre (ou do Segredo Mestre Criptografado)?

g. A mensagem de apresentação criptografada do cliente leva em conta o número de registros SSL?

h. A mensagem de apresentação criptografada do servidor leva em conta o número de registros SSL?

(Captura de tela do Wireshark reimpressa com permissão da Wireshark Foundation.)

P20. Na Seção 8.6.1 mostramos que, sem os números de sequência, Trudy (a mulher do meio) pode causar destruição em uma sessão SSL ao trocar os segmentos TCP. Trudy pode fazer algo semelhante ao excluir um segmento TCP? O que ela precisa fazer para ter sucesso em seu ataque de exclusão? Quais serão os seus efeitos?

P21. Suponha que Alice e Bob estão se comunicando por uma sessão SSL. Imagine que um atacante, que não tem nenhuma das chaves compartilhadas, insira um segmento TCP falso em um fluxo de pacotes com a soma de verificação TCP e números sequenciais corretos (e endereços IP e números de porta corretos). A SSL do lado receptor aceitará o pacote falso e passará a carga útil à aplicação receptora? Por que ou por que não?

P22. As seguintes questões de Verdadeiro/Falso se referem à Figura 8.28.

a. Quando um hospedeiro em 172.16.1/24 envia um datagrama ao servidor Amazon.com, o roteador R1 vai criptografar o datagrama usando IPsec.

b. Quando um hospedeiro em 172.16.1/24 envia um datagrama a um servidor 172.16.2/24, o roteador R1 mudará os endereços de origem e destino do datagrama IP.

c. Suponha que um hospedeiro 172.16.1/24 inicie uma conexão TCP com um servidor Web em 172.16.2/24. Como parte dessa conexão, todos os datagramas enviados pelo R1 terão o número de protocolo 50 no campo esquerdo do cabeçalho IPv4.

d. Considere o envio de um segmento TCP de um hospedeiro em 172.16.1/24 a um hospedeiro em 172.16.2/24. Suponha que o reconhecimento desse segmento se perde, então um TCP reenvia o segmento.

P23. Considere o exemplo na Figura 8.28. Suponha que Trudy é a mulher do meio, que insere datagramas no fluxo de datagramas indo de R1 a R2. Como parte do ataque de repetição, Trudy envia uma cópia duplicada de um dos datagramas enviados de R1 a R2. R2 decriptografará o datagrama duplicado e o encaminhará à rede da filial? Se não, descreva em detalhes como R2 detecta o datagrama duplicado.

P24. Considere o seguinte pseudoprotocolo WEP. A chave é de 4 bits e o IV, de 2 bits. O IV é anexado ao final da chave quando está gerando o fluxo de chaves. Suponha que uma chave secreta compartilhada é 1010. Os fluxos de chaves para quatro entradas possíveis são:

```
101000: 0010101101010101001011010100100...
101001: 1010011011001010110100100101101...
101010: 0001101000111100010100101001111...
101011: 1111101010000000101010100010111...
```

Suponha que todas as mensagens tenham 8 bits. Considere que o ICV (controle de integridade) seja de 4 bits e calculado pelo OU-exclusivo nos primeiros 4 bits de dados com os últimos 4 bits de dados. Suponha que o pseudopacote WEP consista em três campos: primeiro o campo IV, depois o campo de mensagem e por último o campo ICV, com alguns deles criptografados.

a. Queremos enviar a mensagem m = 10100000 usando IV = 11 e WEP. Qual serão os valores nos três campos WEP?

b. Mostre que, quando o receptor decriptografa o pacote WEP, ele recupera a mensagem e o ICV.

c. Suponha que Trudy intercepte um pacote WEP (não necessariamente com IV = 11) e quer modificá-lo antes de encaminhá-lo ao receptor. Considere que Trudy inverta o primeiro bit do ICV. Admitindo que ela não conheça o fluxo de chaves para quaisquer IVs, que outros bits Trudy precisa gerar também, para que os pacotes recebidos passem pela verificação do ICV?

d. Justifique sua resposta modificando os bits do pacote WEP da parte (a), decodificando o pacote resultante e verificando o controle de integridade.

P25. Forneça uma tabela de filtro e uma tabela de conexão para um *firewall* de estado que seja tão restrito quanto possível, mas que efetue o seguinte:

a. Permite que todos os usuários internos estabeleçam sessões Telnet com hospedeiros externos.

b. Permite que usuários externos naveguem pela página Web da empresa em 222.22.0.12.

c. Mas, em qualquer outro caso, bloqueia todo o tráfego interno e externo.

A rede interna é 222.22/16. Em sua solução, suponha que a tabela de conexão esteja normalmente ocultando três conexões, de dentro para fora. Você precisará inventar um endereço IP e número de portas apropriados.

P26. Suponha que Alice queira visitar a página Web activist.com usando um serviço do tipo TOR. Esse serviço usa dois servidores *proxy* colaboradores, Proxy1 e Proxy2. Alice primeiro obtém os certificados (cada um contendo uma chave pública) para Proxy1 e Proxy2 de um servidor central. Indique com $K_1^+(\ )$, $K_2^+(\ )$, $K_1^-(\ )$ e $K_2^-(\ )$ para a criptografia/decriptografia com chaves RSA pública e privada.

a. Usando um diagrama de tempo, forneça um protocolo (o mais simples possível) que permita a Alice estabelecer uma sessão compartilhada de chave S_1 com Proxy1. Indique com $S_1(m)$ a criptografia/decriptografia do dado m com a chave compartilhada S_1.

b. Usando um diagrama de tempo, forneça um protocolo (o mais simples possível) que permita a Alice estabelecer uma sessão compartilhada da chave S_2 com Proxy2, *sem revelar seu endereço IP ao Proxy2.*

c. Suponha que as chaves compartilhadas S_1 e S_2 estejam determinadas. Usando um diagrama de tempo, forneça um protocolo (o mais simples possível e *não usando uma criptografia de chave pública*) que permita a Alice requisitar uma página HTML de activist.com *sem revelar seu endereço IP ao Proxy2* e *sem revelar ao Proxy1 qual site ela está visitando.* Seu diagrama deve terminar com uma requisição HTTP chegando a activist.com.

WIRESHARK LAB

Neste laboratório (disponível no site de apoio), investigamos o protocolo SSL (do inglês *Secure Socket Layer*). Lembre-se de que na Seção 8.6 o SSL é usado para a segurança de uma conexão TCP, e é extensivamente usada na prática para a segurança de transações pela Internet. Neste laboratório, vamos focalizar os registros SSL enviados por uma conexão TCP. Tentaremos delinear e classificar cada registro, focando no entendimento do porquê e como de cada registro. Investigamos os vários tipos de registro SSL, assim como os campos nas mensagens SSL, analisando um relatório dos registros SSL enviados entre seu hospedeiro e um servidor de comércio eletrônico.

IPSEC LAB

Neste laboratório (disponível na página Web), exploraremos como criar SAs IPsec entre caixas Linux. Você pode fazer a primeira parte com duas caixas Linux simples, cada uma com um adaptador Ethernet. Mas, na segunda parte do laboratório, você precisará de quatro caixas Linux, duas tendo dois adaptadores Ethernet. Na segunda metade do laboratório, você criará SAs IPsec usando protocolos ESP no modo túnel. Primeiro você criará as SAs manualmente e depois fazendo o IKE criar as SAs.

ENTREVISTA

Steven M. Bellovin

Steven M. Bellovin ingressou no corpo docente da Universidade de Columbia depois de muitos anos no Network Services Research Lab do AT&T Labs Research, em Florham Park, Nova Jersey. Ele trabalha especificamente com redes e segurança, e com as causas que as tornam incompatíveis. Em 1995, Steven recebeu o Usenix Lifetime Achievement Award por seu trabalho na criação da Usenet, a primeira rede de troca de grupos de bate-papo que ligava dois ou mais computadores e permitia que os usuários compartilhassem informações e se juntassem em discussões. Steve também foi eleito membro da National Academy of Engineering. Obteve bacharelado na Columbia University e doutorado na University of North Carolina, em Chapel Hill.

O que o fez se decidir pela especialização na área de segurança de redes?

O que vou dizer parecerá estranho, mas a resposta é simples. Estudei programação de sistemas e administração de sistemas, o que levou naturalmente à segurança. E sempre me interessei por comunicações, desde a época em que trabalhava em parte do tempo com programação de sistemas, quando ainda estava na universidade.

Meu trabalho na área de segurança continua a ser motivado por duas coisas — um desejo de manter os computadores úteis, o que significa impedir que sua função seja corrompida por atacantes, e um desejo de proteger a privacidade.

Qual era sua visão sobre a Usenet na época em que o senhor a estava desenvolvendo? Qual é sua visão agora?

No início considerávamos a Usenet como um meio para discutir ciência da computação e programação de computadores em todo o país e para utilizar em questões administrativas locais, como recurso de vendas, e assim por diante. Na verdade, minha previsão original era de uma ou duas mensagens por dia, de 50 a 100 sites no máximo. Mas o crescimento real ocorreu em tópicos relacionados a pessoas, incluindo — mas não se limitando a — interações de seres humanos com computadores. Meus grupos de bate-papo preferidos foram, durante muitos anos, coisas como rec.woodworking (marcenaria), bem como sci.crypt (criptografia).

Até certo ponto, as redes de notícias foram substituídas pela Web. Se eu fosse iniciar o projeto hoje, ele seria muito diferente. Mas ainda é um excelente meio para alcançar um público muito amplo interessado no assunto, sem ter de passar por sites Web particulares.

Quais pessoas o inspiraram profissionalmente? De que modo?

O professor Fred Brooks — fundador e primeiro chefe do departamento de ciência da computação da University of North Carolina, em Chapel Hill, gerente da equipe que desenvolveu o IBM S/360 e o OS/360, e autor do livro *The Mythical Man-Month* — exerceu uma tremenda influência sobre minha carreira. Acima de tudo, ele me ensinou observação e compromissos — como considerar problemas no contexto do mundo real (e como o mundo real é muito mais confuso do que qualquer teórico gostaria que fosse), e como equilibrar interesses conflitantes ao elaborar uma solução. Grande parte do trabalho com computadores é engenharia — a arte de fazer os compromissos certos de modo a satisfazer muitos objetivos contraditórios.

Em sua opinião, qual é o futuro das redes e da segurança?

Até agora, grande parte da segurança que temos vem do isolamento. Um *firewall*, por exemplo, funciona impedindo o acesso a certas máquinas e serviços. Mas vivemos em uma época na qual a conectividade é cada vez maior — ficou mais difícil isolar coisas. Pior ainda, nossos sistemas de produção requerem um número muito maior de componentes isolados, interconectados por redes. Garantir a segurança de tudo isso é um de nossos maiores desafios.

Em sua opinião, tem havido grandes avanços na área de segurança? Até onde teremos de ir?

Ao menos do ponto de vista científico, sabemos como fazer criptografia. E isso é uma grande ajuda. Mas a maioria dos problemas de segurança se deve a códigos defeituosos e este é um problema muito mais sério. Na verdade, é o problema mais antigo da ciência da computação que ainda não foi resolvido — e acho que continuará sendo. O desafio é descobrir como manter a segurança de sistemas quando temos de construí-los com componentes inseguros. Hoje já podemos fazer isso no que tange às falhas de hardware; mas podemos fazer o mesmo em relação à segurança?

O senhor pode dar algum conselho aos estudantes sobre a Internet e segurança em redes?

Aprender os mecanismos é a parte fácil. Aprender a "pensar como um paranoico" é mais difícil. Você tem de lembrar que as distribuições de probabilidade não se aplicam — os atacantes podem encontrar e encontrarão condições improváveis. E os detalhes são importantes — e muito!

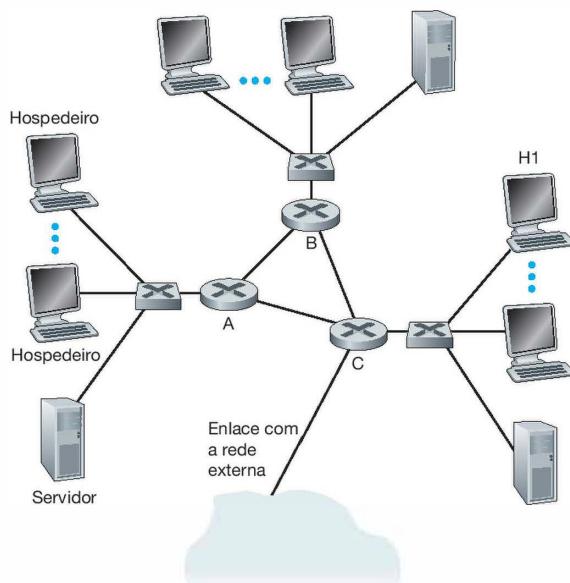
C A P Í T U L O

1 2 3 4 5 6 7 8 9

GERENCIAMENTO DE REDE

Após termos percorrido nosso caminho pelos oito primeiros capítulos deste livro, estamos agora conscientes de que uma rede consiste em muitas peças complexas de hardware e software que interagem umas com as outras — desde os enlaces, comutadores, roteadores, hospedeiros e outros dispositivos, que são os componentes físicos, até os muitos protocolos (tanto em hardware quanto em software) que controlam e coordenam esses componentes. Quando centenas ou milhares de componentes são montados em conjunto por alguma organização para formar uma rede, não é nada surpreendente que por vezes eles apresentem defeitos, que elementos da rede sejam mal configurados, que recursos da rede sejam utilizados em excesso ou que componentes simplesmente "quebrem" (por exemplo, um cabo pode ser cortado, uma latinha de refrigerante pode ser derramada sobre um roteador). O administrador de rede, cuja tarefa é mantê-la "viva e atuante", deve estar habilitado a reagir a esses contratempos (e, melhor ainda, a evitá-los). Com potencialmente milhares de componentes espalhados por uma grande área, ele, em sua central de operações (*network operations center* — NOC), evidentemente necessita de ferramentas que o auxiliem a monitorar, administrar e controlar a rede. Neste capítulo, examinaremos a arquitetura, os protocolos e as bases de informação que um administrador de rede utiliza para realizar essa tarefa.

9.1 O QUE É GERENCIAMENTO DE REDE?

Antes de discutir o gerenciamento de redes em si, vamos considerar alguns cenários ilustrativos do "mundo real" que não são de redes, mas nos quais um sistema complexo com muitos componentes em interação deve ser monitorado, gerenciado e controlado por um administrador. As usinas de geração de energia elétrica têm uma sala de controle, onde mostradores, medidores e lâmpadas monitoram o estado (temperatura, pressão, vazão) de válvulas, tubulações, tanques e outros componentes remotos da instalação industrial. Esses dispositivos permitem que o operador monitore os muitos componentes da planta e podem alertá-lo (o famoso pisca-alerta) caso haja algum problema iminente. O operador responsável executa certas ações para controlar esses componentes. De maneira semelhante, a cabine de um avião é equipada com instrumentos para que o piloto possa monitorar e controlar os muitos componentes de uma aeronave. Nesses dois exemplos, o "administrador" *monitora* equipamentos remotos e *analisa* os dados para garantir que os equipamentos estejam funcionando e operando dentro dos limites especificados (por exemplo, que a fusão do núcleo de uma usina nuclear não esteja na iminência de acontecer ou que o combustível do avião não esteja prestes a acabar), *controla reativamente* o sistema fazendo ajustes de acordo com as modificações ocorridas no sistema ou em seu ambiente e *gerencia proativamente* o sistema (por exemplo, detectando tendências ou comportamentos anômalos que permitem executar uma ação antes

que surjam problemas sérios). De modo semelhante, o administrador de rede vai monitorar, gerenciar e controlar ativamente o sistema do qual está encarregado.

Nos primórdios das redes de computadores, quando elas ainda eram artefatos de pesquisa, e não uma infraestrutura usada por milhões de pessoas por dia, "gerenciamento de rede" era algo de que nunca se tinha ouvido falar. Se alguém descobrisse um problema, poderia realizar alguns testes, como o *ping*, para localizar a fonte do problema e, em seguida, modificar os ajustes do sistema, reiniciar o software ou o hardware ou chamar um colega para fazer isso. (O RFC 789 traz uma discussão de fácil leitura sobre a primeira grande "queda" da ARPAnet em 27 de outubro de 1980, muito antes da existência de ferramentas de gerenciamento, e sobre os esforços realizados para entender os motivos dessa queda e recuperar a rede.) Como a Internet pública e as intranets privadas cresceram e se transformaram de pequenas redes em grandes infraestruturas globais, a necessidade de gerenciar mais sistematicamente a enorme quantidade de componentes de hardware e software nessas redes também se tornou mais importante.

Com o intuito de motivar nosso estudo de gerenciamento de redes, vamos começar com um exemplo simples. A Figura 9.1 ilustra uma pequena rede constituída de três roteadores e alguns hospedeiros e servidores. Mesmo para uma rede tão simples, há muitos cenários em que o administrador muito se beneficiará por ter à mão as ferramentas de gerenciamento adequadas:

- *Detecção de falha em uma placa de interface em um hospedeiro ou roteador*. Com ferramentas de gerenciamento apropriadas, uma entidade de rede (por exemplo, o roteador A) pode indicar ao administrador que uma de suas interfaces não está funcionando. (Isso decerto é melhor do que receber um telefonema, no NOC (centro de operações), de um usuário zangado dizendo que a conexão com a rede caiu!) Um administrador de rede que monitora e analisa de maneira ativa o tráfego pode *realmente* impressionar o usuário (aquele, o zangado) detectando problemas na interface bem antes e substituindo a placa de interface antes que ela caia. Isso poderá ser feito, por exemplo, se o administrador notar um aumento de erros de somas de verificação em quadros que estão sendo enviados por uma placa de interface que está prestes a falhar.
- *Monitoração de hospedeiro*. O administrador de rede pode verificar periodicamente se todos os hospedeiros da rede estão ativos e operacionais. Mais uma vez, ele pode, de fato, impressionar um usuário da rede, reagindo proativamente a um problema (falha em um hospedeiro) antes de o defeito ser relatado pelo usuário.
- *Monitoração de tráfego para auxiliar o oferecimento de recursos*. Um administrador de rede pode monitorar padrões de tráfego entre origens e destinos e notar, por exemplo, que, trocando servidores entre segmentos de LAN, o total de tráfego que passa por várias LANs poderia ser reduzido de maneira significativa. Imagine a felicidade geral com um desempenho melhor sem o custo de novos equipamentos. De modo semelhante, monitorando a utilização de um enlace, um administrador de rede pode determinar que um segmento de LAN ou o enlace com o mundo externo está sobrecarregado e que um enlace de maior largura de banda deve ser providenciado (ainda que com um custo mais alto). Ele também poderia ser alertado automaticamente quando o nível de congestionamento de um enlace ultrapassasse determinado limite, para providenciar um enlace de maior largura de banda antes que o congestionamento se tornasse sério.
- *Detecção de mudanças rápidas em tabelas de roteamento*. A alternância de rotas — mudanças frequentes em tabelas de roteamento — pode indicar instabilidades no roteamento ou um roteador mal configurado. Evidentemente, um administrador de rede que configurou um roteador de modo inapropriado prefere ele mesmo descobrir o erro antes que a rede caia.
- *Monitoração de SLAs*. **Acordos de Nível de Serviços** (*Service Level Agreements* — SLAs) são contratos que definem parâmetros específicos de medida e níveis aceitáveis de desempenho do provedor de rede em relação a essas medidas [Huston, 1999a]. Verizon e Sprint são apenas dois dos muitos provedores de rede que garantem SLAs a seus clientes [AT&T SLA, 2012; Verizon SLA, 2012]. Alguns desses SLAs são: disponibilidade de serviço (interrupção de serviços), latência, vazão e requisitos para notificação da

FIGURA 9.1 UM CENÁRIO SIMPLES QUE ILUSTRA A UTILIZAÇÃO DO GERENCIAMENTO DE REDE

ocorrência de serviço interrompido. É claro que, se um contrato especificar critérios de desempenho de prestação de serviços entre um provedor de rede e seus usuários, então a medição e o gerenciamento do desempenho do sistema também serão de grande importância para o administrador de rede.

- *Detecção de invasão*. Um administrador de rede provavelmente vai querer ser avisado quando chegar tráfego de uma fonte suspeita ou quando se destinar tráfego a ela (por exemplo, hospedeiro ou número de porta). Da mesma maneira, ele talvez queira detectar (e, em muitos casos, filtrar) a existência de certos tipos de tráfego (por exemplo, pacotes roteados pela origem ou um grande número de pacotes SYN dirigidos a determinado hospedeiro) que são característicos de certos tipos de ataque à segurança que consideramos no Capítulo 8.

A International Organization for Standardization (ISO) criou um modelo de gerenciamento de rede que é útil para situar os cenários apresentados em um quadro mais estruturado. São definidas cinco áreas de gerenciamento de rede:

- *Gerenciamento de desempenho*. Sua meta é quantificar, medir, informar, analisar e controlar o desempenho (por exemplo, utilização e vazão) de diferentes componentes da rede. Entre esses componentes estão dispositivos individuais (por exemplo, enlaces, roteadores e hospedeiros), bem como abstrações fim a fim, como um trajeto pela rede. Veremos, em breve, que padrões de protocolo como o SNMP (*Simple Network Management Protocol* — Protocolo Simples de Gerenciamento de Rede) [RFC 3410] desempenham um papel fundamental no gerenciamento de desempenho da Internet.
- *Gerenciamento de falhas*. Seu objetivo é registrar, detectar e reagir às condições de falha da rede. A linha divisória entre gerenciamento de falha e gerenciamento de desempenho é bastante indefinida. Podemos entender o primeiro como o tratamento imediato de falhas transitórias da rede (por exemplo, interrupção de serviço em enlaces, hospedeiros, ou em hardware e software de roteadores), enquanto o segundo adota uma abordagem de longo prazo em relação ao desempenho da rede em face de demandas variáveis de tráfego e falhas ocasionais na rede. Como acontece no gerenciamento de desempenho, o SNMP tem um papel fundamental no gerenciamento de falhas.

- *Gerenciamento de configuração.* O gerenciamento de configuração permite que um administrador de rede saiba quais dispositivos fazem parte da rede administrada e quais são suas configurações de hardware e software. O [RFC 3139] oferece uma visão geral de gerenciamento e requisitos de configuração para redes IP.
- *Gerenciamento de contabilização.* Permite que o administrador da rede especifique, registre e controle o acesso de usuários e dispositivos aos recursos da rede. Quotas de utilização, cobrança por utilização e alocação de acesso privilegiado a recursos fazem parte do gerenciamento de contabilização.
- *Gerenciamento de segurança.* Sua meta é controlar o acesso aos recursos da rede de acordo com alguma política bem definida. As centrais de distribuição de chaves e as autoridades certificadoras que estudamos na Seção 8.3 são componentes do gerenciamento de segurança. O uso de *firewalls* para monitorar e controlar pontos externos de acesso à rede, um tópico que estudamos na Seção 8.9, é outro componente crucial.

Neste capítulo, abordaremos apenas os rudimentos do gerenciamento de rede. Nosso foco é propositalmente limitado — examinaremos apenas a infraestrutura do gerenciamento de rede: a arquitetura geral, os protocolos de gerenciamento de rede e a base de informações que servem para um administrador de rede mantê-la em pé e em funcionamento. *Não* abordaremos os processos de tomada de decisão do administrador de rede, que é quem deve planejar, analisar e reagir às informações gerenciais dirigidas à NOC. Nessa área estão alguns tópicos, como identificação e gerenciamento de falhas [Katzela, 1995; Medhi, 1997; Labovitz, 1997; Steinder, 2002; Feamster, 2005; Wu, 2005; Teixeira, 2006], detecção de anomalias [Lakhina, 2004; Lakhina, 2005; Barford, 2009] e outros mais. Tampouco abordaremos o tópico mais amplo do gerenciamento de serviços [Saydam, 1996; RFC 3052] — o fornecimento de recursos, como largura de banda, capacidade do servidor e outros recursos computacionais e de comunicação necessários para cumprir os requisitos de serviço específicos da missão de uma empresa.

Uma pergunta frequente é: "O que é gerenciamento de rede?". Nossa discussão anterior expôs e ilustrou a necessidade de alguns usos desse gerenciamento. Concluiremos esta seção com uma definição em uma única sentença (embora um tanto longa), dada por Saydam [1996]:

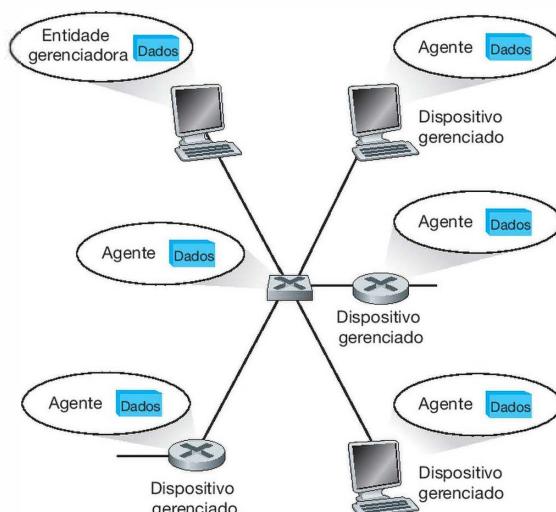
> *Gerenciamento de rede inclui a implementação, a integração e a coordenação de elementos de hardware, software e humanos, para monitorar, testar, consultar, configurar, analisar, avaliar e controlar os recursos da rede, e de elementos, para satisfazer às exigências operacionais, de desempenho e de qualidade de serviço a um custo razoável.*

É uma sentença de perder o fôlego, mas é uma definição viável. Nas seções seguintes, acrescentaremos um pouco de recheio a essa definição bastante enxuta de gerenciamento de rede.

9.2 A INFRAESTRUTURA DO GERENCIAMENTO DE REDE

Vimos, na seção anterior, que o gerenciamento de rede exige a capacidade de "monitorar, testar, consultar, configurar [...] e controlar" os componentes de hardware e software. Como os dispositivos da rede são distribuídos, fazer isso exigirá, no mínimo, que o administrador consiga coletar dados (por exemplo, para a finalidade de monitoração) de uma entidade remota e efetuar mudanças nessa entidade (por exemplo, controle). Nesta seção, uma analogia humana será útil para entender a infraestrutura necessária para gerenciamento de rede.

Imagine que você seja o dirigente de uma grande organização que têm filiais em todo o mundo. É tarefa sua assegurar que as diversas partes de sua organização operem sem percalços. Como você o fará? No mínimo, você coletará dados periodicamente das filiais por meio de relatórios e de várias medições quantitativas de atividade, produtividade e orçamento. De vez em quando (mas não sempre), você será explicitamente notificado da existência de um problema em uma filial; o gerente que quer galgar a escada corporativa (e talvez ficar com seu cargo) pode enviar relatórios não solicitados mostrando que as coisas estão correndo muito bem na filial a seu cargo. Você examinará os relatórios que receber esperando encontrar operações regulares em todos os lugares, mas,

sem dúvida, achará problemas que demandarão sua atenção. Você poderá iniciar um diálogo pessoal com uma de suas filiais problemáticas, coletar mais dados para entender o problema e, então, passar uma ordem executiva ("Faça essa mudança!") ao gerente.

Implícita nesse cenário humano muito comum está uma infraestrutura para controlar a organização — o chefe (você), os locais remotos que estão sendo controlados (as filiais), seus agentes remotos (os gerentes das filiais), protocolos de comunicação (para transmitir relatórios e dados padronizados e para diálogos pessoais) e dados (o conteúdo dos relatórios e as medições quantitativas de atividade, produtividade e orçamento). Cada um desses componentes do gerenciamento organizacional humano tem um correspondente no gerenciamento de rede.

A arquitetura de um sistema de gerenciamento de rede é conceitualmente idêntica a essa analogia simples com uma organização humana. O campo do gerenciamento tem sua terminologia própria para os vários componentes de uma arquitetura de gerenciamento de rede; portanto, adotamos aqui essa terminologia. Como mostra a Figura 9.2, há três componentes principais nessa arquitetura: uma entidade gerenciadora (o chefe, na analogia apresentada), os dispositivos gerenciados (as filiais) e um protocolo de gerenciamento de rede.

A **entidade gerenciadora** é uma aplicação que em geral tem um ser humano no circuito e que é executada em uma estação central de gerenciamento de rede na NOC. Ela é o centro da atividade; ela controla a coleta, o processamento, a análise e/ou a apresentação de informações de gerenciamento de rede. É nela que são iniciadas ações para controlar o comportamento da rede e é aqui que o administrador humano interage com os dispositivos da rede.

Um **dispositivo gerenciado** é um equipamento de rede (incluindo seu software) que reside em uma rede gerenciada. Ele corresponde à filial de nossa analogia humana. Um dispositivo gerenciado pode ser um hospedeiro, um roteador, uma ponte, um *hub*, uma impressora ou um modem. Em seu interior pode haver diversos **objetos gerenciados.** Estes são, na verdade, as peças de hardware propriamente ditas que estão dentro do dispositivo gerenciado (por exemplo, uma placa de interface de rede) e os conjuntos de parâmetros de configuração para as peças de hardware e software (por exemplo, um protocolo de roteamento intradomínio, como o RIP). Em nossa analogia humana, os objetos gerenciados podem ser os departamentos existentes na filial. Esses objetos gerenciados têm informações associadas a eles que são coletadas dentro de uma **Base de Informações de Gerenciamento** (*Management Information Base* — MIB); veremos que os valores dessas in-

FIGURA 9.2 PRINCIPAIS COMPONENTES DE UMA ARQUITETURA DE GERENCIAMENTO DE REDE

formações estão disponíveis para a entidade gerenciadora (e, em muitos casos, podem ser definidos por ela). Em nossa analogia humana, a MIB corresponde aos dados quantitativos (medições de atividade, produtividade e orçamento, podendo este último ser estabelecido pela entidade gerenciadora!) que são trocados entre o escritório central e a filial. Estudaremos as MIBs em detalhes na Seção 9.3. Por fim, reside também em cada dispositivo gerenciado um **agente de gerenciamento de rede**, um processo executado no dispositivo gerenciado, que se comunica com a entidade gerenciadora e que executa ações locais nos dispositivos gerenciados sob o comando e o controle da entidade gerenciadora. O agente de gerenciamento de rede é o gerente da filial em nossa analogia humana.

O terceiro componente da arquitetura é o **protocolo de gerenciamento de rede**. Esse protocolo é executado entre a entidade gerenciadora e o agente de gerenciamento de rede dos dispositivos gerenciados, o que permite que a entidade gerenciadora investigue o estado dos dispositivos gerenciados e, indiretamente, execute ações sobre eles mediante seus agentes. Estes podem usar o protocolo de gerenciamento de rede para informar à entidade gerenciadora a ocorrência de eventos excepcionais (por exemplo, falhas de componentes ou violação de patamares de desempenho). É importante notar que o protocolo de gerenciamento de rede em si não gerencia a rede. Em vez disso, ele fornece uma ferramenta com a qual o administrador pode gerenciar ("monitorar, testar, consultar, configurar, analisar, avaliar e controlar") a rede. Essa é uma distinção sutil, mas importante.

Embora a infraestrutura do gerenciamento de rede seja conceitualmente simples, podemos nos atrapalhar com seu vocabulário especial, como os termos "entidade gerenciadora", "dispositivo gerenciado", "agente de gerenciamento" e "base de informações de gerenciamento". Por exemplo, nessa terminologia, em nosso cenário simples de monitoração de hospedeiros, "agentes de gerenciamento" localizados em "dispositivos gerenciados" são periodicamente examinados pela "entidade gerenciadora" — uma ideia simples, mas um problema linguístico! Mas, com um pouco de sorte, lembrar sempre da analogia com uma organização humana e seus óbvios paralelos com o gerenciamento de rede nos ajudará neste capítulo.

Nossa discussão anterior sobre arquitetura de gerenciamento de rede foi genérica e se aplica, em geral, a vários padrões e esforços que vêm sendo propostos há anos. Os padrões de gerenciamento de rede começaram a amadurecer no final da década de 1980, sendo que o OSI **CMISE/CMIP** (*Common Management Service Element/Common Management Information Protocol* — **Elemento de Serviço de Gerenciamento Comum/ Protocolo de Informação de Gerenciamento Comum**) [Piscatello, 1993; Stallings, 1993; Glitho, 1998] e o **SNMP** (*Simple Network Management Protocol* — **Protocolo Simples de Gerenciamento de Rede**) da Internet [RFC 3410; Stallings, 1999; Rose, 1996] emergiram como os dois padrões mais importantes [Subramanian, 2000]. Ambos foram projetados para ser independentes de produtos ou de redes de fabricantes específicos. Como o SNMP foi projetado e oferecido rapidamente em uma época em que a necessidade de gerenciamento de rede começava a ficar premente, ele encontrou uma ampla aceitação. Hoje, esse protocolo é a estrutura de gerenciamento de rede mais usada e disseminada. Abordaremos o SNMP em detalhes na seção seguinte.

PRINCÍPIOS NA PRÁTICA

A central de operações de rede da Comcast

A rede IP baseada em fibra de classe mundial Comcast oferece produtos e serviços reunidos a 49 milhões de clientes combinados de vídeo, dados e voz. A rede da Comcast inclui mais de 618.000 milhas de rota instalada, 138.000 milhas de rota de fibra, 30.000 milhas de *backbone*, 122.000 nós óticos e armazenamento maciço para a Content Delivery Network da Comcast, que entrega um produto de Video on Demand de mais de 134 Terabytes. Cada parte da rede da Comcast, até as residências e empresas dos

clientes, é monitorada por um dos Centros de Operações da empresa.

A Comcast opera dois Centros Nacionais de Operações de Rede que controlam o *backbone* nacional, redes locais regionais, aplicações nacionais e plataformas específicas que dão suporte à infraestrutura de voz, dados e vídeo para clientes residenciais, comerciais e atacadistas. Além disso, tem três Centros de Operações Divisionais que controlam a infraestrutura local que dá suporte a todos os seus clientes. Os Centros de Operações Nacionais e Divisionais são responsáveis por monitorar, de forma proativa, todos os aspectos de sua rede e o desempenho de produtos todos os dias do ano, 24 horas por dia, utilizando processos e sistemas comuns. Por exemplo, diversos eventos da rede nos níveis nacional e local têm objetivos predefinidos comuns para níveis de severidade, processos de recuperação e tempo médio esperado para restauração. Os centros nacionais e divisionais podem dar apoio uns aos outros se um problema local afetar a operação de um site. Além disso, os Centros de Operações Nacionais e Divisionais têm uma extensa Rede Virtual Privada, que permite aos engenheiros acessarem a rede com segurança para realizar, remotamente, atividades proativas ou reativas de gerenciamento de rede.

A técnica da Comcast para o gerenciamento de rede envolve cinco áreas principais: Gerenciamento de Desempenho, Gerenciamento de Falhas, Gerenciamento de Configuração, Gerenciamento de Contabilização e Gerenciamento de Segurança. **Gerenciamento de Desempenho** focaliza o conhecimento de como a rede/sistemas e aplicações (conhecidos como ecossistema) estão trabalhando com relação a medições predefinidas, específicas da hora do dia, dia da semana ou eventos especiais (por exemplo, tempestades ou eventos importantes, como uma partida de futebol). Essas medições predefinidas existem por todo o caminho do serviço, desde a residência ou empresa do cliente e passando pela rede inteira, bem como os pontos de interface com parceiros e pares. Além disso, transações sintéticas são executadas para garantir a continuidade do sistema. **Gerenciamento de Falhas** é definido como a capacidade de detectar, registrar e entender anomalias que podem afetar os clientes. A Comcast utiliza mecanismos de correlação para determinar corretamente a severidade de um evento e atuar de modo apropriado, eliminando ou remediando potenciais problemas antes que afetem os clientes. **Gerenciamento de Configuração** garante que versões apropriadas de hardware e software sejam distribuídas por todos os elementos do ecossistema. Manter esses elementos em seus níveis "dourados" máximos ajuda a evitar consequências indesejadas. **Gerenciamento de Contabilização** garante que os centros de operações tenham um conhecimento claro da provisão e utilização do ecossistema. Isso é importante especialmente para garantir que, em todo o tempo, os centros de operações tenham a capacidade de redirecionar o tráfego com eficiência. **Gerenciamento de Segurança** cuida para que haja controles apropriados para garantir que o ecossistema esteja protegido de modo eficiente contra acesso indevido.

Essas telas mostram ferramentas que dão suporte a correlação, gerenciamento de patamares e destinos, usadas pelos técnicos da Comcast (Cortesia da Comcast).

Centros de Operações de Rede e o ecossistema que eles apoiam não são estáticos. O pessoal de engenharia e operações está sempre reavaliando as medidas de desempenho e ferramentas predefinidas para garantir que as expectativas dos clientes por excelência operacional sejam atendidas.

9.3 A ESTRUTURA DE GERENCIAMENTO PADRÃO DA INTERNET

Ao contrário do que o nome SNMP (Protocolo Simples de Gerenciamento de Rede) possa sugerir, o gerenciamento de rede na Internet é muito mais do que apenas um protocolo para transportar dados de gerenciamento entre uma entidade gerenciadora e seus agentes, e o SNMP passou a ser muito mais complexo do que sugere a palavra "simples". As raízes da atual estrutura de gerenciamento padrão da Internet (*Internet Standard Management Framework*) remontam ao SGMP (*Simple Gateway Monitoring Protocol* — protocolo de monitoramento do *gateway* simples) [RFC 1028]. O SGMP foi projetado por um grupo de pesquisadores, usuários e administradores universitários de rede, cuja experiência com esse protocolo permitiu que eles projetassem, implementassem e oferecessem o SNMP em poucos meses [Lynch, 1993] — um feito muito distante dos processos de padronização atuais, que são bastante prolongados. Desde então, o SNMP evoluiu do SNMPv1 para o SNMPv2 e chegou à sua versão mais recente, o SNMPv3 [RFC 3410], lançada em abril de 1999 e atualizada em dezembro de 2002.

Na descrição de qualquer estrutura para gerenciamento de rede, certas questões devem inevitavelmente ser abordadas:

- O que está sendo monitorado (de um ponto de vista semântico)? E que tipo de controle pode ser exercido pelo administrador de rede?
- Qual é o modelo específico das informações que serão relatadas e/ou trocadas?
- Qual é o protocolo de comunicação para trocar essas informações?

Lembre-se de nossa analogia humana apresentada na seção anterior. O chefe e os gerentes das filiais precisarão acertar entre eles as medições de atividade, produtividade e orçamento que serão usadas para relatar a situação das filiais. De maneira semelhante, eles terão de concordar sobre que tipos de ações o chefe poderá realizar (por exemplo, cortar o orçamento, ordenar que o gerente da filial modifique alguma característica da operação do escritório ou demitir o pessoal e fechar a filial). Em um maior nível de detalhamento, eles precisarão concordar sobre o modo como esses dados serão relatados. Por exemplo, em que moeda (dólares, reais?) será apresentado o relatório de orçamento? Em que unidades será medida a produtividade? Embora talvez pareçam triviais, esses detalhes terão de ser acertados. Por fim, a maneira pela qual a informação trafegará entre o escritório central e as filiais (isto é, o protocolo de comunicação) deve ser especificada.

A estrutura de gerenciamento-padrão da Internet aborda essas questões. Ela é constituída de quatro partes:

- Definições dos *objetos de gerenciamento de rede*, conhecidos como objetos MIB. Na Estrutura de Gerenciamento de Padrão da Internet, as informações de gerenciamento são representadas como uma coletânea de objetos gerenciados que, juntos, formam um banco de informações virtuais, conhecido como MIB (*Management Information Base*). Um objeto MIB pode ser um contador, tal como o número de datagramas IP descartados em um roteador por causa de erros em cabeçalhos de datagramas IP ou o número de erros de detecção de portadora em uma placa de interface Ethernet; um conjunto de informações descritivas, como a versão do software que está sendo executado em um servidor DNS; informações de estado, como se um determinado dispositivo está funcionando corretamente; ou informações específicas sobre protocolos, como um caminho de roteamento até um destino. Assim, os objetos MIB definem as informações de gerenciamento mantidas por um dispositivo gerenciado. Objetos MIB relacionados são reunidos em **módulos MIB**. Em nossa analogia com uma organização humana, a MIB define a informação transportada entre a filial e a sede.
- Uma *linguagem de definição de dados*, conhecida como SMI (*Structure of Management Information* — Estrutura de Informação de Gerenciamento), que define os tipos de dados, um modelo de objeto e regras para escrever e revisar informações de gerenciamento. Objetos MIB são especificados nessa linguagem de definição de dados. Em nossa analogia humana, a SMI é usada para definir os detalhes do formato das informações que serão trocadas.

- *Um protocolo, SNMP*. O SNMP é usado para transmitir informações e comandos entre uma entidade gerenciadora e um agente que os executa em nome da entidade dentro de um dispositivo de rede gerenciado.
- *Capacidades de segurança e de administração*. A adição dessas capacidades representa o aprimoramento mais importante do SNMPv3 em comparação com o SNMPv2.

Assim, a arquitetura de gerenciamento da Internet é modular por projeto, com uma linguagem de definição de dados e de MIB independente de protocolo e um protocolo independente de MIB. O interessante é que essa arquitetura modular foi criada primeiro para facilitar a transição de um gerenciamento de rede baseado em SNMP para uma estrutura de gerenciamento de rede que estava sendo desenvolvida pela ISO, que era a arquitetura de gerenciamento que concorria com o SNMP quando este foi projetado — uma transição que nunca aconteceu. Com o tempo, contudo, a modularidade do projeto do SNMP permitiu que ele evoluísse mediante três importantes revisões, com cada uma das quatro partes do SNMP já discutidas antes se desenvolvendo de modo independente. Ficou bem claro que a decisão pela modularidade foi correta, ainda que tenha sido tomada pela razão errada!

Nas subseções seguintes, examinaremos com mais detalhes os quatro componentes mais importantes da estrutura de gerenciamento-padrão da Internet.

9.3.1 SMI (Estrutura de Informações de Gerenciamento)

A **Estrutura de Informações de Gerenciamento** (*Structure of Management Information* — SMI) (um componente da estrutura de gerenciamento de rede cujo nome é bastante estranho e não dá nenhuma pista quanto à sua funcionalidade) é a linguagem usada para definir as informações de gerenciamento que residem em uma entidade gerenciada de rede. Essa linguagem de definição é necessária para assegurar que a sintaxe e a semântica dos dados de gerenciamento de rede sejam bem definidas e não apresentem ambiguidade. Note que a SMI não define uma instância específica para os dados em uma entidade gerenciada de rede, mas a linguagem na qual a informação está especificada. Os documentos que descrevem a SMI para o SNMPv3 (confusamente denominada SMIv2) são [RFC 2578; RFC 2579; RFC 2580]. Vamos examinar a SMI de baixo para cima, começando com seus tipos de dados básicos. Em seguida, veremos como os objetos gerenciados são descritos em SMI e, então, como os objetos gerenciados relacionados entre si são agrupados em módulos.

Tipos de dados básicos da SMI

O RFC 2578 especifica os tipos de dados básicos da linguagem SMI de definição de módulos MIB. Embora a SMI seja baseada na linguagem de definição de objetos ASN.1 (*Abstract Syntax Notation One* — notação de sintaxe abstrata 1) [ISO X.680, 2002] (ver Seção 9.4), tantos foram os tipos de dados específicos da SMI acrescentados que esta deve ser considerada uma linguagem de definição de dados por direito próprio. Os 11 tipos de dados básicos definidos no RFC 2578 aparecem na Tabela 9.1. Além desses objetos escalares, também é possível impor uma estrutura tabular sobre um conjunto ordenado de objetos MIB usando a construção SEQUENCE OF; consulte o RFC 2578 para mais detalhes. Grande parte dos tipos de dados da Tabela 9.1 será familiar (ou autoexplicativa) para a maioria dos leitores. O único tipo de dado que discutiremos com mais detalhes será o OBJECT IDENTIFIER, usado para dar nome a um objeto.

Construções SMI de nível mais alto

Além dos tipos de dados básicos, a linguagem de definição de dados SMI também fornece construções de linguagem de nível mais alto.

TABELA 9.1 TIPOS DE DADOS BÁSICOS DA SMI

Tipo de dado	Descrição
INTEGER	Número inteiro de 32 bits, como definido em ASN.1, com valor entre -2^{31} e $2^{31}-1$, inclusive, ou um valor de uma lista de valores constantes possíveis, nomeados.
Integer32	Número inteiro de 32 bits, com valor entre -2^{31} e $2^{31}-1$, inclusive.
Unsigned32	Número inteiro de 32 bits sem sinal na faixa de 0 a $2^{32}-1$, inclusive.
OCTET STRING	Cadeia de bytes de formato ASN.1 que representa dados binários arbitrários ou de texto de até 65.535 bytes de comprimento.
OBJECT IDENTIFIER	Formato ASN.1 atribuído administrativamente (nome estruturado); veja a Seção 9.3.2.
IPaddress	Endereço Internet de 32 bits, na ordem de bytes da rede.
Counter32	Contador de 32 bits que cresce de 0 a $2^{32}-1$ e volta a 0.
Counter64	Contador de 64 bits.
Gauge32	Número inteiro de 32 bits que não faz contagens além de $2^{32}-1$ nem diminui para menos do que 0.
TimeTicks	Tempo, medido em centésimos de segundo, transcorrido a partir de algum evento.
Opaque	Cadeia ASN.1 não interpretada, necessária por compatibilidade com versões anteriores.

A construção OBJECT-TYPE é utilizada para especificar o tipo de dado, o status e a semântica de um objeto gerenciado. Esses objetos gerenciados contêm os dados de gerenciamento que estão no núcleo do gerenciamento de rede. Há mais de dez mil objetos definidos em diversos RFCs da Internet [RFC 3410]. A construção OBJECT-TYPE tem quatro cláusulas. A cláusula SYNTAX de uma definição OBJECT-TYPE especifica o tipo de dado básico associado ao objeto. A cláusula MAX-ACCESS especifica se o objeto gerenciado pode ser lido, escrito, criado ou ter seu valor incluído em uma notificação. A cláusula STATUS indica se a definição do objeto é atual e válida, obsoleta (caso em que não deve ser executada, pois sua definição está incluída por motivos históricos apenas) ou desaprovada (obsoleta, mas implementável por causa de sua interoperabilidade com implementações mais antigas). A cláusula DESCRIPTION contém uma definição textual e legível do objeto; ela "documenta" a finalidade do objeto gerenciado e deve fornecer todas as informações semânticas necessárias para executá-lo.

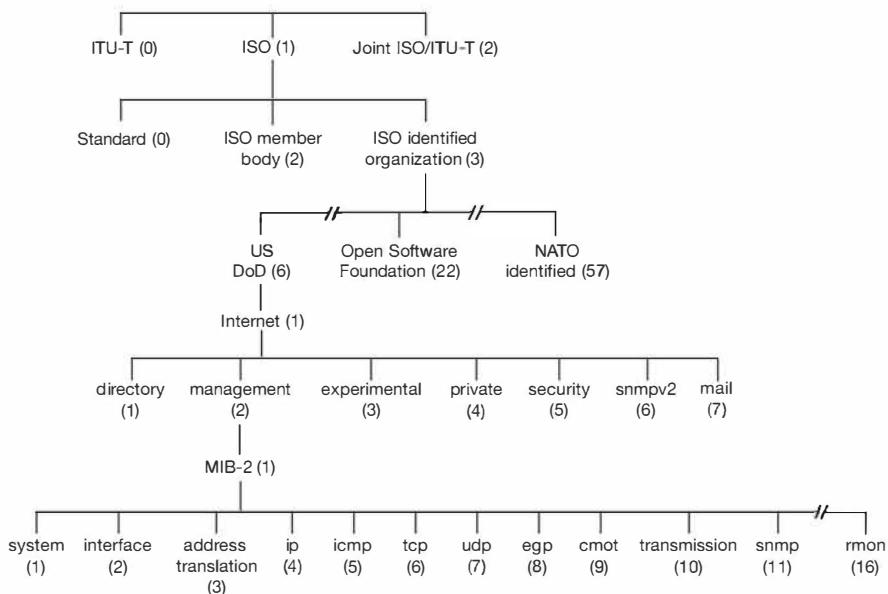
Como exemplo de uma construção OBJECT-TYPE, considere a definição do tipo de objeto `ipSystemStatsInDelivers` do RFC 4293. Esse objeto define um contador de 32 bits que monitora o número de datagramas IP recebidos no dispositivo gerenciado e entregues com sucesso a um protocolo de camada superior. A última linha dessa definição diz respeito ao nome desse objeto, um tópico que consideraremos na subseção seguinte.

```
ipSystemStatsInDelivers OBJECT-TYPE
      SYNTAX Counter32
      MAX-ACCESS read-only
      STATUS current
      DESCRIPTION
            "The total number of datagrams successfully
            delivered to IPuser-protocols (including ICMP).

            When tracking interface statistics, the counter
            of the interface to which these datagrams were
            addressed is incremented. This interface might
            not be the same as the input interface for
            some of the datagrams.

            Discontinuities in the value of this counter can
            occur at re-initialization of the management
            system, and at other times as indicated by the
            value of ipSystemStatsDiscontinuityTime."
      ::= { ipSystemStatsEntry 18 }
```

A construção MODULE-IDENTITY permite que objetos relacionados entre si sejam agrupados, como conjunto, dentro de um "módulo". Por exemplo, o RFC 4293 especifica o módulo MIB que define objetos gerenciados (incluindo `ipSystemStatsInDelivers`) para gerenciar implementações do IP e de seu protocolo associado, o ICMP. O [RFC 4022] especifica o módulo MIB para TCP, e o [RFC 4133] especifica o módulo MIB para UDP. O [RFC 4502] define o módulo MIB para monitoração remota, RMON. Além de conter as definições OBJECT-TYPE dos objetos gerenciados dentro do módulo, a construção MODULE-IDENTITY contém cláusulas para documentar informações de contato do autor do módulo, a data da última atualização, um histórico de revisões e uma descrição textual do módulo. Como exemplo, considere o módulo de definição para gerenciamento do protocolo IP:

```
ipMIB MODULE-IDENTITY
      LAST-UPDATED "200602020000Z"
      ORGANIZATION "IETF IPv6 MIB Revision Team"
      CONTACT-INFO
            "Editor:
            Shawn A. Routhier
            Interworking Labs
            108 Whispering Pines Dr. Suite 235
            Scotts Valley, CA 95066
            USA
            EMail: <sar@iwl.com>"
      DESCRIPTION
            "The MIB module for managing IP and ICMP
            implementations, but excluding their
            management of IP routes.

            Copyright (C) The Internet Society (2006).
            This version of this MIB module is part of
            RFC 4293; see the RFC itself for full legal
            notices."

      REVISION "200602020000Z"
      DESCRIPTION
            "The IP version neutral revision with added
            IPv6 objects for ND, default routers, and
            router advertisements. As well as being the
            successor to RFC 2011, this MIB is also the
            successor to RFCs 2465 and 2466. Published
            as RFC 4293."

      REVISION "199411010000Z"
      DESCRIPTION
            "A separate MIB module (IP-MIB) for IP and
            ICMP management objects. Published as RFC
            2011."

      REVISION "199103310000Z"
      DESCRIPTION
            "The initial revision of this MIB module was
            part of MIB-II, which was published as RFC
            1213."
      ::= { mib-2 48}
```

A construção NOTIFICATION-TYPE é usada para especificar informações referentes a mensagens SNMPv2 Trap e InformationRequest geradas por um agente ou por uma entidade gerenciadora; veja a

Seção 9.3.3. Entre essas informações estão um texto de descrição (DESCRIPTION) que especifica quando tais mensagens devem ser enviadas, bem como uma lista de valores que devem ser incluídos na mensagem gerada; veja o [RFC 2578] para obter mais detalhes. A construção MODULE-COMPLIANCE define o conjunto de objetos gerenciados dentro de um módulo que um agente deve implementar. A construção AGENT-CAPABILITIES especifica as capacidades dos agentes relativas às definições de notificação de objetos e de eventos.

9.3.2 Base de informações de gerenciamento: MIB

Como já dissemos antes, a **Base de Informações de Gerenciamento** (*Management Information Base* — MIB) pode ser imaginada como um banco virtual de informações que guarda objetos gerenciados cujos valores, coletivamente, refletem o "estado" atual da rede. Esses valores podem ser consultados e/ou definidos por uma entidade gerenciadora por meio do envio de mensagens SNMP ao agente que está rodando em um dispositivo gerenciado em nome da entidade gerenciadora. Objetos gerenciados são especificados utilizando a construção OBJECT-TYPE da SMI discutida anteriormente e agrupados em módulos MIB utilizando a construção MODULE-IDENTITY.

A IETF tem estado muito atarefada com a padronização de módulos MIB associados a roteadores, hospedeiros e outros equipamentos de rede, o que inclui dados básicos de identificação sobre determinado componente de hardware e informações de gerenciamento sobre as interfaces e os protocolos de dispositivos da rede. Até 2006 havia mais de duzentos módulos MIB baseados em padrões e um número ainda maior de módulos MIB especificados por fabricantes privados. Com todos esses padrões, a IETF precisava encontrar um modo de identificar e dar nome aos módulos padronizados, bem como aos objetos gerenciados específicos dentro de um módulo. Em vez de começar do nada, a IETF adotou uma estrutura padronizada de identificação de objetos (nomeação) que já tinha sido publicada pela ISO. Como acontece com muitas entidades dedicadas à padronização, a ISO tinha "grandes planos" para sua estrutura padronizada de identificação de objetos — identificar todo e qualquer objeto padronizado possível (por exemplo, formato de dados, protocolo ou informação) em qualquer rede, independentemente das organizações dedicadas à padronização das redes (por exemplo, IETF, ISO, IEEE ou ANSI), do fabricante do equipamento ou do proprietário da rede. Um objetivo bem grandioso mesmo! A estrutura de identificação de objeto adotada pela ISO é parte da linguagem de definição de objetos ASN.1 [ISO X.680, 2002], que discutiremos na Seção 9.4. Os módulos MIB padronizados já têm seu próprio cantinho confortável dentro dessa estrutura de nomeação bastante abrangente, como veremos a seguir.

Como mostra a Figura 9.3, pela estrutura de nomeação da ISO, os objetos são nomeados de modo hierárquico. Note que cada ponto de ramo da árvore tem um nome e um número (entre parênteses); assim, qualquer ponto da árvore pode ser identificado pela sequência de nomes ou números que especificam o trajeto da raiz até aquele ponto na árvore identificadora. Um programa baseado na Web — divertido, mas incompleto e não oficial — que percorre parte da árvore de identificadores de objetos (usando informações sobre os ramos, oferecidas por voluntários) pode ser encontrado em OID Repository [2012].

No topo da hierarquia estão a ISO e o ITU-T, as duas principais entidades de padronização que tratam da ASN.1, bem como um ramo para o esforço conjunto realizado por essas duas organizações. No ramo ISO da árvore, encontramos registros para todos os padrões ISO (1.0) e para os emitidos por entidades padronizadoras de vários países membros da ISO (1.2). Embora não apareça na Figura 9.3, logo abaixo desse ramo da árvore (ISO/países membros, também conhecido como 1.2), encontraríamos os Estados Unidos (1.2.840), embaixo dos quais encontraríamos um número para os padrões IEEE e ANSI e para os padrões específicos de empresas privadas. Entre essas empresas, estão a RSA (1.2.840.11359) e a Microsoft (1.2.840.113556), sob a qual encontraríamos os Microsoft File Formats (1.2.840.113556.4) para vários produtos da Microsoft,

FIGURA 9.3 ÁRVORE DE IDENTIFICADORES DE OBJETOS ASN.1

como o Word (1.2.840.113556.4.2). Mas estamos interessados em redes (e *não* nos arquivos do Microsoft Word), portanto, vamos voltar nossa atenção ao ramo denominado 1.3, os padrões emitidos por entidades reconhecidas pela ISO. Entre elas estão o Departamento de Defesa dos Estados Unidos (6) (sob o qual encontraremos os padrões da Internet), a Open Software Foundation (22), a associação de empresas aéreas SITA (69) e as entidades identificadas pela OTAN (57), além de muitas outras organizações.

Sob o ramo `Internet` da árvore (1.3.6.1), há sete categorias. Sob o ramo `private` (1.3.6.1.4), encontramos uma lista [IANA, 2009b] dos nomes e códigos de empresas privadas para as mais de quatro mil empresas privadas que se registraram na Internet Assigned Numbers Authority (IANA) [IANA, 2009a]. Sob o ramo `management` (1.3.6.1.2) e `MIB-2` (1.3.6.1.2.1), encontramos a definição dos módulos MIB padronizados. Puxa! É uma longa jornada até chegarmos ao nosso cantinho no espaço de nomes da ISO!

Módulos MIB padronizados

O ramo mais baixo da árvore da Figura 9.3 mostra alguns dos módulos MIB importantes, orientados para hardware (`system` e `interface`), bem como os associados a alguns dos protocolos mais importantes da Internet. O RFC 5000 relaciona todos os módulos MIB padronizados. Embora os RFCs referentes às MIBs sejam uma leitura tediosa e difícil, é muito instrutivo (isto é, assim como comer verduras, "é bom para você") considerar algumas definições de módulos MIB para ter uma ideia do tipo de informação que há em um módulo.

Os objetos gerenciados que ficam sob o título system contêm informações gerais sobre o dispositivo que está sendo gerenciado; todos os dispositivos gerenciados devem suportar os objetos MIB do grupo system. A Tabela 9.2 define os objetos gerenciados no grupo system, de acordo com o [RFC 1213]. A Tabela 9.3 define os objetos gerenciados no módulo MIB para o protocolo UDP em uma entidade gerenciada.

TABELA 9.2 OBJETOS GERENCIADOS NO GRUPO SYSTEM DA MIB-2

Identificador de objeto	Nome	Tipo	Descrição (segundo RFC 1213)
1.3.6.1.2.1.1.1	`sysDescr`	`OCTET STRING`	"Nome completo e identificação da versão do tipo de hardware do sistema, do sistema operacional do software e do software de rede."
1.3.6.1.2.1.1.2	`sysObjectID`	`OBJECT IDENTIFIER`	ID atribuído pelo fabricante do objeto fornece um meio fácil e não ambíguo para determinar que tipo de objeto está sendo gerenciado.
1.3.6.1.2.1.1.3	`sysUpTime`	`TimeTicks`	"O tempo (em centésimos de segundo) desde que a parte de gerenciamento de rede do sistema foi reinicializada pela última vez."
1.3.6.1.2.1.1.4	`sysContact`	`OCTET STRING`	"A pessoa de contato para esse nó gerenciado, juntamente com a informação sobre como contatá-la."
1.3.6.1.2.1.1.5	`sysName`	`OCTET STRING`	"Um nome atribuído administrativamente para esse nó gerenciado. Por convenção, esse é o nome de domínio totalmente qualificado do nó."
1.3.6.1.2.1.1.6	`sysLocation`	`OCTET STRING`	"A localização física do nó."
1.3.6.1.2.1.1.7	`sysServices`	`Integer32`	Um valor codificado que indica o conjunto de serviços disponível no nó: aplicações físicas (por exemplo, um repetidor), de enlace/sub-rede (por exemplo, ponte), de Internet (por exemplo, gateway IP), fim a fim (por exemplo, hospedeiro).

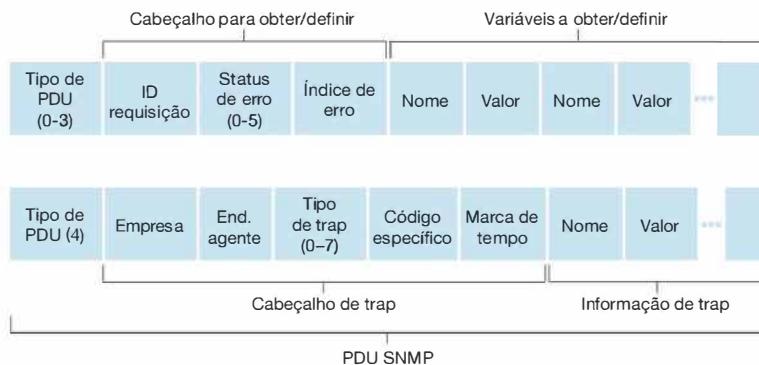
TABELA 9.3 OBJETOS GERENCIADOS NO MÓDULO MIB-2 UDP

Identificador de objeto	Nome	Tipo	Descrição (segundo RFC 4113)
1.3.6.1.2.1.7.1	`udpInDatagrams`	`Counter32`	"Número total de datagramas UDP entregues a usuários UDP."
1.3.6.1.2.1.7.2	`udpNoPorts`	`Counter32`	"Número total de datagramas UDP recebidos para os quais não havia nenhuma aplicação na porta de destino."
1.3.6.1.2.1.7.3	`udpInErrors`	`Counter32`	"Número de datagramas UDP recebidos que não puderam ser entregues por outras razões que não a falta de uma aplicação na porta de destino."
1.3.6.1.2.1.7.4	`udpOutDatagrams`	`Counter32`	"Número total de datagramas UDP enviados dessa entidade."

9.3.3 Operações do protocolo SNMP e mapeamentos de transporte

O Protocolo simples de gerenciamento de rede versão 2 (SNMPv2) [RFC 3416] é usado para transportar informações da MIB entre entidades gerenciadoras e agentes, executando em nome das entidades gerenciadoras. A utilização mais comum do SNMP é em um **modo comando-resposta**, no qual a entidade gerenciadora SNMPv2 envia uma requisição a um agente SNMPv2, que a recebe, realiza alguma ação e envia uma resposta à requisição. Em geral, uma requisição é usada para consultar (recuperar) ou modificar (definir) valores de

objetos MIB associados a um dispositivo gerenciado. Um segundo uso comum do SNMP é para um agente enviar uma mensagem não solicitada, conhecida como **mensagem *trap***, à entidade gerenciadora. As mensagens *trap* são usadas para notificar uma entidade gerenciadora de uma situação excepcional que resultou em mudança nos valores dos objetos MIB. Vimos antes, na Seção 9.1, que o administrador de rede pode querer receber uma mensagem *trap*, por exemplo, quando uma interface cai, quando o congestionamento atinge um nível predefinido em um enlace ou quando ocorre qualquer outro evento importante. Observe que há uma série de compromissos importantes entre consulta de objetos (interação comando-resposta) e *trapping*; veja os exercícios ao final deste capítulo.

O SNMPv2 define sete tipos de mensagens, conhecidas genericamente como PDUs (*protocol data units* – Protocolo de unidade de dados), conforme apresentado na Tabela 9.4 e descrito em seguida. O formato da PDU pode ser visto na Figura 9.4.

- As PDUs `GetRequest`, `GetNextRequest` e `GetBulkRequest` são enviadas de uma entidade gerenciadora a um agente para requisitar o valor de um ou mais objetos MIB no dispositivo gerenciado do agente. Os identificadores de objeto dos objetos MIB cujos valores estão sendo requisitados são especificados na parte de vinculação de variáveis da PDU. `GetRequest`, `GetNextRequest` e `GetBulkRequest` diferem no grau de especificidade de seus pedidos de dados. `GetRequest` pode requisitar um conjunto arbitrário de valores MIB; múltiplas `GetNextRequest` podem ser usadas para percorrer a sequência de uma lista ou tabela de objetos MIB, e `GetBulkRequest` permite que um grande bloco de dados seja devolvido, evitando a sobrecarga incorrida quando tiverem de ser enviadas múltiplas mensagens `GetRequest` ou `GetNextRequest`. Em todos os três casos, o agente responde com uma PDU `Response` que contém os identificadores de objetos e seus valores associados.
- A PDU `SetRequest` é usada por uma entidade gerenciadora para estabelecer o valor de um ou mais objetos MIB em um dispositivo gerenciado. Um agente responde com uma PDU `Response` que contém uma mensagem de estado de erro "noError" para confirmar que o valor realmente foi estabelecido.
- A PDU `InformRequest` é usada por uma entidade gerenciadora para comunicar a outra entidade gerenciadora informações MIB remotas à entidade receptora. A entidade receptora responde com uma PDU `Response` com a mensagem de estado de erro "noError" para reconhecer o recebimento da PDU `InformRequest`.
- O tipo final de PDU SNMPv2 é a mensagem *trap*. Mensagens *trap* são geradas assincronamente, isto é, *não* são geradas em resposta a uma requisição recebida, mas em resposta a um evento para o qual a entidade gerenciadora requer notificação. O RFC 3418 define tipos conhecidos de *trap* que incluem uma partida a frio ou a quente realizada por um dispositivo, a ativação ou interrupção de um enlace, a perda de um vizinho ou um evento de falha de autenticação. Uma requisição de *trap* recebida não exige resposta de uma entidade gerenciadora.

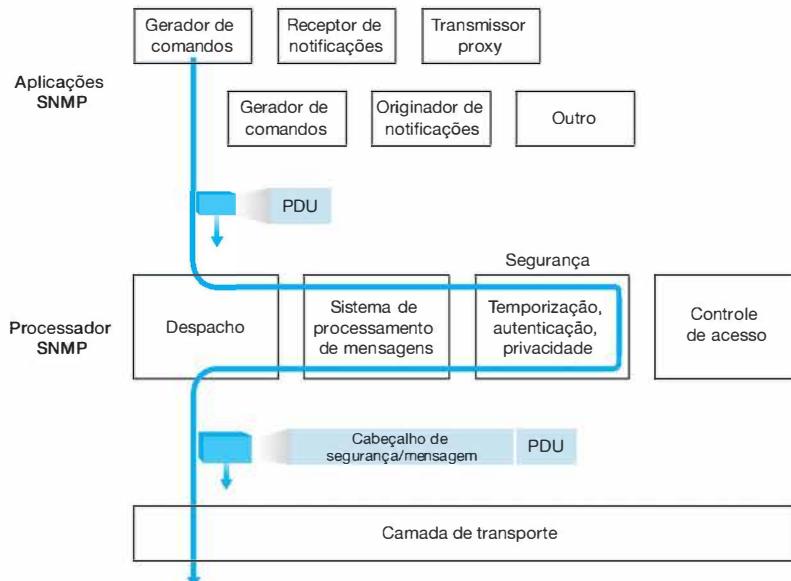
Sabendo da natureza comando-resposta do SNMPv2, convém observar que, embora as SNMP-PDUs possam ser transportadas por muitos protocolos de transporte diferentes, elas normalmente são transportadas na carga útil de um datagrama UDP. Na verdade, o RFC 3417 estabelece que o UDP é o "mapeamento de transporte preferencial". Já que o UDP é um protocolo de transporte não confiável, não há garantia de que um comando ou sua resposta será recebido no destino pretendido. O campo *request* ID da PDU é usado pela entidade gerenciadora para numerar as requisições que faz a um agente; a resposta de um agente adota a *request* ID daquela do comando recebido. Assim, o campo *request* ID pode ser usado pela entidade gerenciadora para detectar comandos ou respostas perdidos. Cabe à entidade gerenciadora decidir se retransmitirá um comando se nenhuma resposta correspondente for recebida após determinado período de tempo. Em particular, o padrão SNMP não impõe nenhum procedimento específico de retransmissão, nem mesmo diz que o comando deve ser enviado em primeiro lugar. Ele requer apenas que a entidade gerenciadora "aja com responsabilidade em relação à frequência e à duração das retransmissões". Isso, é claro, nos leva a pensar como deve agir um protocolo "responsável"!

TABELA 9.4 TIPOS DE PDU SNMPv2

Tipo de SNMPv2-PDU	Remetente-receptor	Descrição
`GetRequest`	gerente a agente	pega valor de uma ou mais instâncias de objetos MIB
`GetNextRequest`	gerente a agente	pega valor da próxima instância de objeto MIB na lista ou tabela
`GetBulkRequest`	gerente a agente	pega valores em grandes blocos de dados, por exemplo, valores em uma grande tabela
`InformRequest`	gerente a gerente	informa à entidade gerenciadora remota valores da MIB que são remotos para seu acesso
`SetRequest`	gerente a agente	define valores de uma ou mais instâncias de objetos MIB
`Response`	agente a gerente ou gerente a gerente	gerado em resposta a `GetRequest`, `GetNextRequest`, `GetBulkRequest`, `SetRequest PDU`, ou `InformRequest`
`SNMPv2-Trap`	agente a gerente	informa ao gerente um evento excepcional

FIGURA 9.4 FORMATO DA PDU SNMP

9.3.4 Segurança e administração

Os projetistas do SNMPv3 têm dito que o "SNMPv3 pode ser considerado um SNMPv2 com capacidades adicionais de segurança e de administração" [RFC 3410]. Decerto, há mudanças no SNMPv3 em relação ao SNMPv2, mas em nenhum lugar elas são mais evidentes do que nas áreas da administração e da segurança. O papel central da segurança no SNMPv3 era de particular importância, já que a falta de segurança adequada resultava no uso do SNMP primordialmente para monitorar, em vez de controlar (por exemplo, `SetRequest` é pouquíssimo usada no SNMPv1).

À medida que o SNMP amadurecia, passando por três versões, sua funcionalidade crescia, mas infelizmente crescia também o número de documentos de padronização relacionados a ele. Isso fica evidenciado pelo fato de que há agora um RFC [RFC 3411] que "descreve uma arquitetura para descrever os Ambientes de Gerenciamento do SNMP"! Embora a ideia de uma "arquitetura" para "descrever um ambiente" possa ser um pouco excessiva para nossa cabeça, o objetivo do RFC 3411 é admirável — introduzir uma linguagem comum para descrever a funcionalidade e as ações executadas por um agente ou entidade gerenciadora SNMPv3. A arquitetura de uma entidade SNMPv3 é direta, e viajar por ela servirá para solidificar nosso entendimento do SNMP.

As denominadas **aplicações SNMP** consistem em um gerador de comandos, um receptor de notificações e um transmissor *proxy* (todos normalmente encontrados em uma entidade gerenciadora); um elemento respondedor de comandos e um originador de notificações (ambos encontrados em geral em um agente), e na possibilidade de outras aplicações. O gerador de comandos gera as PDUs `GetRequest`, `GetNextRequest`, `GetBulkRequest` e `SetRequest`, que examinamos na Seção 9.3.3, e processa as respostas recebidas dessas PDUs. O respondedor de comandos executa em um agente e recebe, processa e responde (usando a mensagem `Response`) às PDUs `GetRequest`, `GetNextRequest`, `GetBulkRequest` e `SetRequest` recebidas. A aplicação originadora de notificações de um agente gera PDUs `Trap`; essas PDUs podem ser recebidas e processadas em uma aplicação receptora de notificações em uma entidade gerenciadora. A aplicação do transmissor *proxy* repassa as PDUs de requisição, notificação e resposta.

Uma PDU enviada por uma **aplicação SNMP** passa, em seguida, por um "processador" SNMP, antes de ser enviada via protocolo de transporte apropriado. A Figura 9.5 mostra como uma PDU gerada pela aplicação geradora de comandos entra primeiro no módulo de despacho, onde é determinada a versão do SNMP. A PDU é então processada no sistema de processamento de mensagens, no qual é envolvida em um cabeçalho de mensagem que contém o número da versão do SNMP, uma ID de mensagem e informações sobre o tamanho desta. Caso seja necessária criptografia ou autenticação, são incluídos também os campos de cabeçalho apropriados para essas informações; veja mais detalhes no [RFC 3411]. Por fim, a mensagem SNMP (a PDU gerada pela aplicação e mais as informações do cabeçalho de mensagem) é passada ao protocolo de transporte apropriado. O protocolo de transporte preferencial para transportar mensagens SNMP é o UDP (isto é, as mensagens SNMP são transportadas como carga útil de um datagrama UDP), e o número de porta preferencial para o SNMP é a porta 161. A porta 162 é usada para mensagens trap.

Vimos antes que as mensagens SNMP são usadas não só para monitorar, mas também para controlar (por exemplo, por meio do comando `SetRequest`) elementos da rede. É claro que um intruso que conseguisse interceptar mensagens SNMP e/ou gerar seus próprios pacotes SNMP na infraestrutura de gerenciamento poderia criar um grande tumulto na rede. Assim, é crucial que mensagens SNMP sejam transmitidas com segurança. Surpreendentemente, foi apenas na versão mais recente do SNMP que a segurança recebeu a atenção merecida.

FIGURA 9.5 PROCESSADOR E APLICAÇÕES DO SNMPv3

A segurança SNMPv3 é conhecida como **segurança baseada no usuário** [RFC 3414], pois utiliza o conceito tradicional de um usuário, identificado por um nome de usuário, ao qual as informações de segurança — uma senha, um valor de chave ou acessos privilegiados — são associadas. O SNMPv3 fornece criptografia, autenticação, proteção contra ataques de reprodução (veja a Seção 8.3) e controle de acesso.

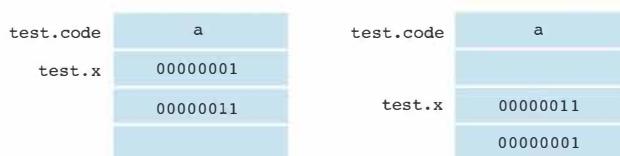
- *Criptografia*. As PDUs SNMP podem ser criptografadas usando o DES (Criptografia de dados padrão) no modo Encadeamento de Blocos de Cifras (CBC). Note que, como o DES é um sistema de chaves compartilhadas, o valor da chave secreta dos dados de codificação do usuário deve ser conhecido pela entidade receptora que deve decodificar os dados.
- *Autenticação*. O SNMP usa a técnica de MAC (*Message Authentication Code*) que estudamos na Seção 8.3.1 para prover proteção e autenticação contra adulteração [RFC 4301]. Lembre-se de que um MAC necessita que um emissor e um receptor conheçam uma chave secreta comum.
- *Proteção contra ataques de reprodução*. Lembre-se de que podem ser usados *nonces* (veja o Capítulo 8) para proteção contra ataques de reprodução. O SNMPv3 adota uma técnica relacionada. Para garantir que uma mensagem recebida não seja uma reprodução de alguma mensagem anterior, o receptor exige que o remetente inclua em cada mensagem um valor baseado em um contador no *receptor*. Esse contador, que funciona como um *nonce*, reflete o período de tempo decorrido entre a última reinicialização do software de gerenciamento de rede do remetente e o número total de reinicializações desde a última vez que o software de gerenciamento de rede do receptor foi configurado. Contanto que o contador em uma mensagem recebida esteja dentro de uma determinada margem de erro em relação ao próprio valor do receptor, a mensagem é aceita como uma mensagem original e a partir daí pode ser autenticada e/ou decriptada. Consulte o [RFC 3414] para obter mais detalhes.
- *Controle de acesso*. O SNMPv3 fornece um controle de acesso baseado em vistas [RFC 3415] que controla quais das informações de gerenciamento de rede podem ser consultadas e/ou definidas por quais usuários. Uma entidade SNMP armazena informações sobre direitos de acesso e políticas em um banco de dados de configuração local (*Local Configuration Datastore* — LCD). Partes do LCD são acessíveis elas próprias como objetos gerenciados, definidos na MIB de Configuração do Modelo de Controle de Acesso Baseado em Vistas (*View-Based Access Control Model Configuration*) [RFC 3415] e, portanto, podem ser gerenciados e manipulados remotamente via SNMP.

9.4 ASN.1

Neste livro, examinamos uma série de assuntos interessantes sobre redes de computadores. Esta seção sobre a ASN.1, contudo, pode não fazer parte da lista dos dez tópicos mais interessantes. Assim como as verduras, conhecer a ASN.1 e a questão mais ampla de serviços de apresentação é algo que "é bom para você". A ASN.1 é um padrão originado na ISO, usado em uma série de protocolos relacionados à Internet, em particular na área de gerenciamento de rede. Por exemplo, vimos na Seção 9.3 que as variáveis MIB do SNMP estão inextricavelmente ligadas à ASN.1. Portanto, embora o material sobre a ASN.1 apresentado nesta seção seja bastante árido, esperamos que o leitor nos dê um voto de confiança e acredite que o material é importante.

Para motivar nossa discussão, faça o seguinte exercício mental: suponha que fosse possível copiar dados, de maneira confiável, da memória de um computador diretamente para a de outro computador. Se isso fosse possível, o problema da comunicação estaria "resolvido"? A resposta a essa pergunta depende de como se define "problema de comunicação". Com certeza, uma cópia perfeita, memória a memória, comunicaria com exatidão os bits e os bytes de uma máquina para outra. Mas essa cópia exata de bits e bytes significa que, quando o software que estiver sendo executado no computador receptor acessar esses dados, ele verá os mesmos valores que estavam armazenados na memória do computador remetente? A resposta é "não necessariamente"! O ponto crucial da questão é que diferentes arquiteturas de computadores, diferentes sistemas operacionais e compiladores têm diferentes convenções de armazenamento e representação de dados. Quando se trata de comunicar e armazenar

dados entre vários computadores (como acontece em todas as redes de comunicação), fica evidente que esse problema da representação de dados tem de ser resolvido.

Como exemplo desse problema, considere o fragmento simples em linguagem C a seguir. Como essa estrutura deveria ser disposta na memória?

```
struct {
       char code;
       int x;
       } test;
       test.x = 259;
       test.code = 'a';
```

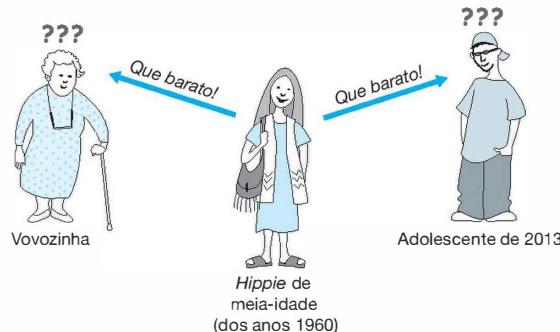
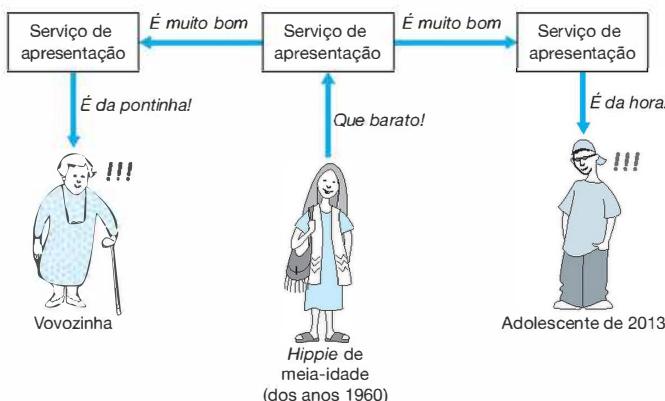
O lado esquerdo da Figura 9.6 mostra uma disposição possível para esses dados em uma arquitetura hipotética: há um único byte de memória que contém o caractere "a", seguido de uma palavra de 16 bits que contém o valor inteiro 259, armazenado com o byte mais significativo antes. A disposição na memória de outro computador é mostrada no lado direito da Figura 9.6. O caractere "a" é seguido pelo valor inteiro armazenado com o byte menos significativo antes e com o inteiro de 16 bits alinhado para começar em um limite de palavra de 16 bits. Certamente, se quiséssemos executar uma cópia ao pé da letra entre as memórias desses dois computadores e usar a mesma definição de estrutura para acessar os valores armazenados, veríamos resultados diferentes nos dois computadores!

O fato de diferentes arquiteturas terem diferentes formatos para os dados internos é um problema real e universal. O problema específico do armazenamento de inteiros em diferentes formatos é tão comum que tem até um nome. A ordem de armazenamento de inteiros "*big-endian*" armazena primeiro os bytes mais significativos (nos endereços de armazenamento mais baixos). A ordem "*little-endian*" armazena primeiro os bytes menos significativos. Os processadores Sparc da Sun e os da Motorola são do tipo "*big-endian*", ao passo que os da Intel são do tipo "*little-endian*". Como curiosidade, os termos "*big-endian*" e "*little-endian*" vêm do livro *Viagens de Gulliver*, de Jonathan Swift, no qual dois grupos de pessoas insistem, dogmaticamente, em fazer uma coisa simples de duas maneiras diferentes (esperamos que a analogia com a comunidade da arquitetura de computadores fique clara). Um grupo da Terra de Lilliput insiste em quebrar os ovos pela extremidade maior (os "*big-endians*"), enquanto o outro grupo insiste em quebrá-los pela extremidade menor. Essa diferença foi a causa de um grande conflito e de uma rebelião civil.

Sabendo que diferentes computadores armazenam e representam dados de modos diferentes, como os protocolos de rede devem enfrentar o problema? Por exemplo, se um agente SNMP estiver prestes a enviar uma mensagem Response que contém um número inteiro que representa a contagem do número de datagramas UDP recebidos, como ele deveria representar o valor inteiro a ser enviado à entidade gerenciadora — na ordem "*big-endian*" ou na ordem "*little-endian*"? Uma alternativa seria o agente enviar os bytes do inteiro na mesma ordem em que serão armazenados na entidade gerenciadora. Outra seria o agente enviar o inteiro em sua própria ordem de armazenamento, deixando à entidade gerenciadora a responsabilidade de reordenar os bytes quando necessário. Qualquer uma das alternativas exigiria que o remetente ou o receptor conhecesse o formato de representação de inteiros do outro.

FIGURA 9.6 DUAS ORGANIZAÇÕES DIFERENTES DE DADOS EM DUAS ARQUITETURAS DIFERENTES

test.code	a
test.x	00000001
	00000011

test.code	a
test.x	00000011
	00000001

Uma terceira alternativa é ter um método independente de máquina, de sistema operacional e de linguagem para descrever números inteiros e outros tipos de dados (isto é, uma linguagem de definição de dados) e regras que estabeleçam a maneira como cada um desses tipos de dados deve ser transmitido pela rede. Quando forem recebidos dados de determinado tipo, eles estarão em um formato conhecido e, assim, poderão ser armazenados em qualquer formato específico que um dado computador exija. Tanto a SMI, que estudamos na Seção 9.3, quanto a ASN.1 adotam essa terceira alternativa. No jargão da ISO, os dois padrões descrevem um **serviço de apresentação** — o serviço de transmitir e traduzir informações de um formato específico de uma máquina para outro. A Figura 9.7 ilustra um problema de apresentação no mundo real; nenhum dos receptores entende a ideia essencial que está sendo comunicada — que o interlocutor gosta de algo. Como mostrado na Figura 9.8, um serviço de apresentação pode resolver esse problema traduzindo a ideia para uma linguagem inteligível (pelo serviço de apresentação), independentemente de quem fala, enviando essa informação ao receptor e, em seguida, traduzindo-a para uma linguagem que o receptor entende.

A Tabela 9.5 mostra alguns tipos de dados definidos pela ASN.1. Lembre-se de que encontramos os tipos de dados INTEGER, OCTET STRING e OBJECT IDENTIFIER em nosso estudo anterior da SMI. Como nossa meta aqui (felizmente) não é fornecer uma introdução completa à ASN.1, remetemos o leitor aos padrões ou ao livro impresso e on-line de Larmouth [1996], para a descrição de tipos e construtores ASN.1, como SEQUENCE e SET, que permitem definir os tipos de dados estruturados.

Além de fornecer uma linguagem de definição de dados, a ASN.1 oferece **Regras Básicas de Codificação** *(Basic Encoding Rules* — BERs), que especificam como instâncias de objetos que foram definidas usando a lingua-

FIGURA 9.7 O PROBLEMA DA APRESENTAÇÃO

FIGURA 9.8 SOLUÇÃO DO PROBLEMA DA APRESENTAÇÃO

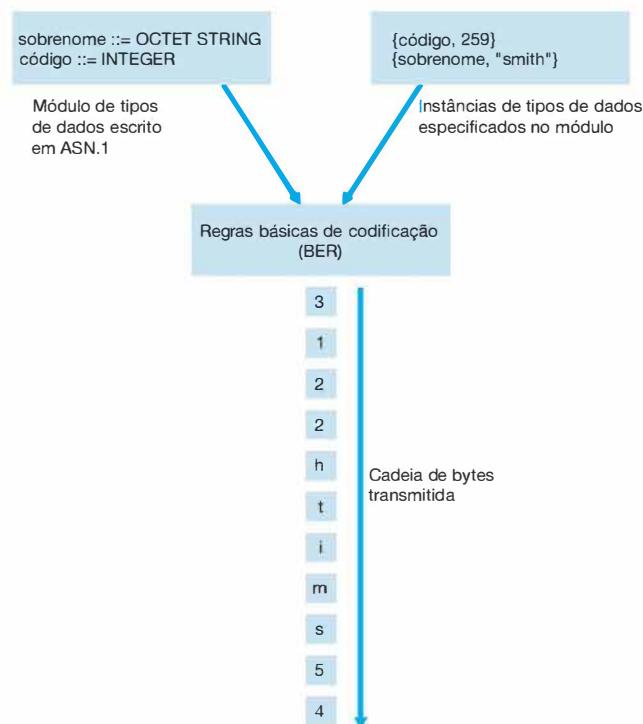



TABELA 9.5 TIPOS DE DADOS ASN.1 SELECIONADOS

Tag	Tipo	Descrição
1	`BOOLEAN`	valor é "verdadeiro" ou "falso"
2	`INTEGER`	pode ser arbitrariamente grande
3	`BITSTRING`	lista de um ou mais bits
4	`OCTET STRING`	lista de um ou mais bytes
5	`NULL`	sem valor
6	`OBJECT IDENTIFIER`	nome, na árvore de nomeação padrão ASN.1; veja Seção 9.2.2
9	`REAL`	ponto flutuante

gem de descrição de dados ASN.1 devem ser enviadas pela rede. A BER adota a abordagem **TLV** (*Type, Length, Value* — **Tipo, Comprimento, Valor**) para a codificação de dados para transmissão. Para cada item de dados a ser remetido, são enviados o tipo dos dados, o comprimento do item de dados e o valor do item de dados, nessa ordem. Com essa simples convenção, os dados recebidos basicamente se autoidentificam.

A Figura 9.9 mostra como os dois itens de dados de um exemplo simples seriam enviados. Nesse exemplo, o remetente quer enviar a cadeia de caracteres "smith" seguida de um valor decimal 259 (que é igual a 00000001 00000011 em linguagem binária, ou a um valor de byte 1 seguido de um valor de byte 3) adotando a ordem

FIGURA 9.9 EXEMPLO DE CODIFICAÇÃO BER

"*big-endian*". O primeiro byte da cadeia transmitida tem o valor 4, que indica que o tipo do item de dado seguinte é um OCTET STRING; esse é o "T" do código TLV. O segundo byte da cadeia contém o comprimento do OCTET STRING, nesse caso, 5. O terceiro byte da cadeia transmitida inicia o OCTET STRING de comprimento 5; ele contém a representação ASCII da letra "s". Os valores T, L e V dos dados seguintes são 2 (o valor de *tag* do tipo INTEGER), 2 (isto é, um inteiro de comprimento de 2 bytes) e a representação "*big-endian*" de 2 bytes do valor decimal 259.

Em nossa discussão, abordamos apenas um subconjunto pequeno e simples da ASN.1. Entre os recursos para aprender mais sobre a ASN.1 estão os documentos padronizados da ASN.1 [ISO X.680, 2002], o livro on-line de Larmouth [2012], relativo ao modelo OSI, e os sites relativos ao ASN.1 [OSS, 2012] e OID Repository [2012].

9.5 CONCLUSÃO

Nosso estudo sobre o gerenciamento de redes — e, na verdade, sobre toda a rede — agora está completo! Neste capítulo final, começamos apresentando os motivos da necessidade de fornecer ferramentas adequadas ao administrador — a pessoa que tem a tarefa de manter a rede "ligada e funcionando" — para monitorar, testar, consultar, configurar, analisar, avaliar e controlar a operação da rede. Nossas analogias com a administração de sistemas complexos, como usinas elétricas, aviões e organizações humanas, ajudaram-nos a fornecer motivos para essa necessidade. Vimos que a arquitetura do sistema de gerenciamento de rede gira em torno de cinco componentes fundamentais: (1) um gerenciador de rede, (2) um conjunto de dispositivos gerenciados remotamente (pelo gerenciador de rede), (3) as bases de informações de gerenciamento (MIBs) existentes nesses dispositivos, contendo dados sobre seu estado e sua operação, (4) os agentes remotos que reportam informação das MIBs e executam ações sob o controle do gerenciador de rede e (5) um protocolo para a comunicação entre o gerenciador de rede e os dispositivos remotos.

Em seguida, examinamos em detalhes a estrutura de gerenciamento de rede da Internet e, em particular, o protocolo SNMP. Vimos como o SNMP apresenta os cinco componentes fundamentais de uma arquitetura de gerenciamento de padrão da Internet e gastamos um bom tempo examinando objetos MIB, a SMI — a linguagem de definição de dados para especificação das MIBs — e o protocolo SNMP em si. Observamos que a SMI e a ASN.1 estão inextricavelmente interligadas e que a ASN.1 desempenha um papel fundamental na camada de apresentação do modelo de referência de sete camadas ISO/OSI, e então fizemos um estudo rápido da ASN.1. Talvez mais importante do que os detalhes da ASN.1 em si foi a necessidade percebida de fornecer a tradução entre formatos de dados específicos de cada computador de uma rede. Embora algumas arquiteturas de rede reconheçam explicitamente a importância desse serviço, por terem uma camada de apresentação, essa camada não existe na pilha de protocolos da Internet.

Convém também observar que há muitos tópicos no gerenciamento de rede que preferimos *não* abordar — tópicos como as falhas de identificação e gerenciamento, a detecção proativa de anomalias, a correlação entre os alarmes e os aspectos mais amplos do gerenciamento de serviço (por exemplo, de forma oposta ao gerenciamento de rede). Embora sejam importantes, esses tópicos merecem um livro dedicado a eles. O leitor pode consultar as referências apresentadas na Seção 9.1.

EXERCÍCIOS DE FIXAÇÃO E PERGUNTAS

Questões de revisão do Capítulo 9

SEÇÃO 9.1

R1. Por que um administrador de rede necessita de ferramentas de gerenciamento de rede? Descreva cinco cenários.

R2. Quais são as cinco áreas de gerenciamento de rede definidas pela ISO?

R3. Qual a diferença entre gerenciamento de rede e gerenciamento de serviço?

SEÇÃO 9.2

R4. Defina os seguintes termos: entidade gerenciadora, dispositivo gerenciado, agente de gerenciamento, MIB e protocolo de gerenciamento de rede.

SEÇÃO 9.3

R5. Qual é o papel da SMI no gerenciamento de rede?

R6. Cite uma diferença importante entre uma mensagem de comando-resposta e uma mensagem *trap* no SNMP.

R7. Quais são os sete tipos de mensagens usados no SNMP?

R8. O que significa um "processador do SNMP"?

SEÇÃO 9.4

R9. Qual é a finalidade da árvore de identificadores de objetos ASN.1?

R10. Qual é o papel da ASN.1 na camada de apresentação nos modelos de referência ISO/OSI?

R11. A Internet tem uma camada de apresentação? Se não tiver, como são tratadas as questões de diferenças entre arquiteturas de máquinas — por exemplo, a representação diferente de números inteiros em máquinas diferentes?

R12. O que significa codificação TLV?

PROBLEMAS

P1. Considere as duas maneiras pelas quais ocorrem as comunicações entre uma entidade gerenciadora e um dispositivo gerenciado: modo comando-resposta e *trapping*. Quais são os prós e os contras dessas duas técnicas, em termos de (1) sobrecarga, (2) tempo de notificação quando ocorrem eventos excepcionais e (3) robustez quanto às mensagens perdidas entre a entidade gerenciadora e o dispositivo gerenciado?

P2. Na Seção 9.3 vimos que era preferível transportar mensagens SNMP em datagramas UDP não confiáveis. Em sua opinião, por que os projetistas do SNMP preferiram o UDP ao TCP como protocolo de transporte para o SNMP?

P3. Qual é o identificador de objeto ASN.1 para o protocolo ICMP (veja a Figura 9.3)?

P4. Suponha que você trabalhe para uma empresa com sede nos Estados Unidos que quer desenvolver sua própria MIB para o gerenciamento de uma linha de produtos. Em que lugar da árvore de identificadores de objetos (veja a Figura 9.3) esse produto seria registrado? (*Dica*: você deve recorrer a alguns RFCs e a outros documentos similares para responder a essa pergunta.)

P5. Lembre-se da Seção 9.3.2, que uma empresa privada (empreendimento) pode criar suas próprias variáveis MIB sob o ramo privado 1.3.6.1.4. Suponha que a IBM quisesse criar uma MIB para seu software do servidor Web. Qual seria o próximo qualificador OID após 1.3.6.1.4? (Para responder a essa questão, você precisará consultar IANA [2009b].) Pesquise na Web para tentar descobrir se essa MIB existe para um servidor da IBM.

P6. Em sua opinião, por que o comprimento precede o valor em uma codificação TLV (ao invés de vir após o valor)?

P7. Considere a Figura 9.9. Qual seria a codificação BER para {`peso`, 75} {`sobrenome`, "`Marco`"}?

P8. Considere a Figura 9.9. Qual seria a codificação BER para {`peso`, 65} {`sobrenome`, "`Dario`"}?

ENTREVISTA

Jennifer Rexford

Jennifer Rexford é professora no departamento de Ciência da Computação da Princeton University. Sua pesquisa tem o amplo objetivo de tornar as redes de computadores mais fáceis de projetar e administrar, com ênfase particular nos protocolos de roteamento. De 1996 a 2004, foi membro do departamento de Gerenciamento e Desempenho de Redes da AT&T Labs-Research. Enquanto estava na AT&T, projetou técnicas e ferramentas para medição de rede, engenharia de tráfego e configuração de roteadores, que foram implementadas na rede de *backbone* da AT&T. Jennifer é coautora do livro *Web Protocols and Practice: Networking Protocols, Caching, and Traffic Measurement*, publicado pela Addison-Wesley em maio de 2001. Foi presidente da ACM SIGCOMM de 2003 a 2007. Graduou-se como bacharel em engenharia elétrica pela Princeton University em 1991 e obteve mestrado e doutorado em engenharia elétrica e ciência da computação pela Universidade de Michigan em 1993 e 1996, respectivamente. Em 2004, Jennifer foi vencedora do Grace Murray Hopper Award da ACM como jovem profissional de destaque em computação, aparecendo na lista TR-100 do MIT dos inovadores com idade menor que 35 anos.

Por favor, descreva um ou dois dos projetos mais interessantes em que você já trabalhou durante sua carreira. Quais foram os maiores desafios?

Quando eu era pesquisadora na AT&T, um grupo nosso projetou uma nova forma de gerenciar o roteamento nas redes de *backbone* dos ISPs. Em geral, os operadores de rede configuram cada roteador individualmente, e esses roteadores executam protocolos distribuídos para calcular caminhos através da rede. Acreditamos que o gerenciamento de rede seria mais simples e mais flexível se os operadores da rede pudessem exercer controle *direto* sobre o modo como os roteadores repassam o tráfego com base em uma visão *em nível de rede* da topologia e do tráfego. A plataforma de controle de roteamento (RCP — *Routing Control Platform*) que projetamos e montamos poderia calcular as rotas para todo o *backbone* da AT&T em um único computador comercial, e poderia controlar roteadores legados sem modificação. Para mim, esse projeto foi interessante porque tivemos uma ideia provocadora, um sistema funcional e por fim uma execução real em uma rede operacional.

Que mudanças e inovações você antecipa que acontecerão no gerenciamento de redes no futuro?

Em vez de apenas "aparafusar" o gerenciamento de rede em cima das redes existentes, os pesquisadores e profissionais estão começando a projetar redes que são fundamentalmente mais fáceis de gerenciar. Como nosso trabalho inicial no RCP, a ideia principal nas chamadas redes definidas por software (SDN — *Software Defined Networking*) é executar um controlador que possa instalar regras de tratamento de pacotes de baixo nível nos comutadores subjacentes, usando um protocolo padrão. Esse controlador pode executar diversas aplicações de gerenciamento de rede, como o controle de acesso dinâmico, mobilidade transparente do usuário, engenharia de tráfego, balanceamento de carga do servidor, uso eficiente das redes em termos de energia e assim por diante. Acredito que a SDN é uma grande oportunidade para acertar o gerenciamento de rede, repensando a relação entre os dispositivos de rede e o software que os controla.

Como você vê o futuro das redes e da Internet?

As redes compõem um campo muito interessante, pois as aplicações e as tecnologias utilizadas mudam o tempo todo. Estamos sempre reinventando! Quem teria previsto, há apenas cinco ou dez anos, o domínio dos smartphones, permitindo que usuários móveis acessem aplicações existentes e também novos serviços baseados em sua localização? O surgimento da computação em nuvem está fundamentalmente mudando a relação entre os usuários e as aplicações que eles executam, e os senso-

res em rede estão permitindo diversas aplicações novas. O ritmo da inovação é mesmo inspirador.

A rede de apoio é um componente decisivo em todas essas inovações. Mesmo assim, a rede está notoriamente "no caminho" — limitando o desempenho, comprometendo a confiabilidade, restringindo aplicações e complicando a implementação e o gerenciamento de serviços. Devemos lutar para tornar a rede do futuro tão invisível quanto o ar que respiramos, de modo que nunca fique no caminho de novas ideias e serviços valiosos. Para isso, precisamos elevar o nível de abstração acima dos dispositivos de rede e protocolos individuais (e seus respectivos acrônimos!), de modo que possamos raciocinar sobre a rede como um todo.

Que pessoas a inspiraram profissionalmente?

Há muito tempo tenho me inspirado em Sally Floyd, do International Computer Science Institute. Sua pesquisa é sempre significativa, focalizando os desafios importantes enfrentados pela Internet. Ela mergulha a fundo em questões difíceis, até que entenda completamente o problema e o espaço das soluções, e dedica muita energia para "fazer as coisas acontecerem", como empurrar suas ideias para padrões de protocolos e equipamentos de rede. Além disso, ela oferece retorno à comunidade, através de serviços profissionais em diversas organizações de padrões e pesquisa, e também criando ferramentas (como os simuladores ns-2 e ns-3 bastante utilizados), que permitem que outros pesquisadores tenham sucesso. Ela se aposentou em 2009, mas sua influência no campo será sentida durante anos.

Quais são suas recomendações para alunos que desejam seguir carreira em ciência da computação e redes?

Redes é um campo inerentemente interdisciplinar. A aplicação de técnicas de outras disciplinas aos problemas de rede é uma ótima maneira de levar o campo adiante. Temos visto inovações tremendas no uso de redes, vindas de áreas tão diversificadas quanto teoria de filas, teorias de jogos, teoria de controle, sistemas distribuídos, otimização de redes, linguagens de programação, aprendizado de máquina, algoritmos, estruturas de dados e assim por diante. Creio que familiarizar-se com um campo relacionado, ou colaborar de perto com especialistas nessas áreas, seja um modo maravilhoso de preparar um alicerce mais forte para as redes, de modo que possamos aprender como montar redes que sejam dignas de confiança da sociedade. Além das disciplinas teóricas, o campo das redes é interessante porque criamos artefatos reais, que pessoas reais utilizam. Dominar o modo como projetamos e montamos sistemas — ganhando experiência em sistemas operacionais, arquitetura de computador etc. — é outro modo fantástico de ampliar seus conhecimentos em redes, ajudando a mudar o mundo.

REFERÊNCIAS

Uma nota sobre URLs. Nas referências a seguir, fornecemos URLs para páginas Web, documentos apenas da Web e outros materiais que não foram publicados em conferências ou periódicos (quando pudemos localizar um URL para tal material). Não fornecemos URLs para publicações de configuração e periódico, pois esses documentos normalmente podem ser localizados por um mecanismo de busca, pelo site da conferência (por exemplo, artigos em todas as conferências e seminários *ACM SIGCOMM* podem ser localizados por http://www.acm.org/sigcomm), ou por meio de uma assinatura de biblioteca digital. Embora todos os URLs fornecidos a seguir fossem válidos (e testados) em janeiro de 2012, URLs podem se tornar desatualizados. Por favor, consulte a versão *on-line* deste livro (http://www.awl.com/kurose-ross) para obter uma bibliografia atualizada.

Uma nota sobre Solicitações de Comentários (RFCs) da Internet: Cópias de RFCs da Internet estão disponíveis em muitos sites. O RFC Editor da Internet Society (o órgão que supervisiona as RFCs) mantém o site, em <http://www.rfc-editor.org>. Esse site lhe permite procurar um RFC específico por título, número ou autores, e mostrará atualizações de quaisquer RFCs listadas. As RFCs da Internet podem ser atualizadas ou podem se tornar obsoletas por outras RFCs mais recentes. Nosso site favorito para obter RFCs é a fonte original — <http://www.rfc-editor.org>.

3COM ADDRESSING 2012. "White paper: Understanding IP addressing: Everything you ever wanted to know". <http://www.3com.com/other/pdfs/infra/corpinfo/en_US/501302.pdf>.

3GPP 2012. Third Generation Partnership Project homepage, <http://www.3gp.org/>.

3GPP NETWORK ARCHITECTURE 2012. 3GPP, "TS 23.002: Network Architecture: Digital Cellular Telecommunications System (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE". <http://www.3gp.org/ftp/Specs/html-info/23002.htm>.

ALBITZ P. & LIU C. *DNS and BIND.* Petaluma,: O'Reilly & Associates, 1993.

ABRAMSON, N. "The Aloha System—Another Alternative for Computer Communications". *Proc. 1970 Fall Joint Computer Conference, AFIPS Conference,* p. 37, 1970.

ABRAMSON, N. "Development of the Alohanet". *IEEE Transactions on Information Theory,* v.IT-31, n.3 (mar. 1985), p. 119–123.

ABRAMSON, N. "The Alohanet – Surfing for Wireless Data". *IEEE Communications Magazine,* v.47, n.12, p. 21–25.

ABU-LIBDEH, H.; COSTA, P.; ROWSTRON, A.; O'SHEA, G.; DONNELLY, A. "Symbiotic Routing in Future Data Centers". *Proc. 2010 ACM SIGCOMM.*

ADHIKARI, V· K.; JAIN, S.; CHEN, Y.; ZHANG, Z. L. "Vivisecting YouTube: An Active Measurement Study". *Technical Report*, University of Minnesota, 2011.

ADHIKARI, V. K.; GAO, Y.; HAO, F.; VARVELLO, M.; HILT, V.; STEINER, M.; ZHANG, Z. L. "Unreeling Netflix: Understanding and Improving Multi-CDN Movie Delivery". *Technical Report*, University of Minnesota, 2012.

AFANASYEV, A.; TILLEY, N.; REIHER, P.; KLEINROCK, L. "Host-to-Host Congestion Control for TCP". *IEEE Communications Surveys & Tutorials*, v.12, n.3, p. 304–342.

AGARWAL, S.; LORCH, J. "Matchmaking for Online Games and Other Latency-sensitive P2P Systems". *Proc. 2009 ACM SIGCOMM*.

AHN, J. S.; DANZIG, P. B.; LIU, Z.; YAN, Y. "Experience with TCP Vegas: Emulation and Experiment". *Proc. 1995 ACM SIGCOMM* (Boston, ago. 1995), p. 185–195.

AKAMAI homepage, http://www.akamai.com.

AKELLA, A.; SESHAN, S.; SHAIKH, A. "An Empirical Evaluation of Wide-Area Internet Bottlenecks". *Proc. 2003 ACM Internet Measurement Conference* (nov. 2003).

AKHSHABI, S.; BEGEN, A. C.; DOVROLIS, C. "An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP". *Proc. 2011 ACM Multimedia Systems Conf.*

AKYILDIZ, I.; GUTIERREX-ESTEVEZ, D.; REYES, E. "The Evolution to 4G Cellular Systems, LTE Advanced". *Physical Communication*. Elsevier, 3 (2010), 217–244.

ALCATEL-LUCENT. "Introduction to Evolved Packet Core". <http://downloads.lightreading.com/wplib/alcatellucent/ALU_WP_Intro_to_EPC.pdf>.

AL-FARES, M.; LOUKISSAS, A.; VAHDAT, A. "A Scalable, Commodity Data Center Network Architecture". *Proc. 2008 ACM SIGCOMM*.

ALIZADEH, M.; GREENBERG, A.; MALTZ, D.; PADHYE, J.; PATEL, P.; PRABHAKAR, B.; SENGUPTA, S.; SRIDHARAN, M. "Data Center TCP (DCTCP)". *Proc. 2010 ACM SIGCOMM*.

ALLMAN, E. "The Robustness Principle Reconsidered: Seeking a Middle Ground". *Communications of the ACM*, v.54, n.8 (ago. 2011), p. 40–45.

ANDERSEN, J. B.; RAPPAPORT, T. S.; YOSHIDA, S. "Propagation Measurements and Models for Wireless Communications Channels". *IEEE Communications Magazine*, (jan.1995), p. 42–49.

ANDREWS, M.; SHEPHERD, M.; SRINIVASAN, A.; WINKLER, P.; ZANE, F. "Clustering and Server Election Using Passive Monitoring". *Proc. 2002 IEEE INFOCOM*.

ANDROUTSELLIS-THEOTOKIS, S.; SPINELLIS, D. "A Survey of Peer-to-Peer Content Distribution Technologies". *ACM Computing Surveys*, v.36, n.4 (dez. 2004), p. 335–371.

APERJIS, C.; FREEDMAN, M. J.; JOHARI, R. "Peer-Assisted Content Distribution with Prices". *Proc. ACM CoNEXT'08* (Madri, dez. 2008).

APPENZELLER, G.; KESLASSY, I.; MCKEOWN, N. "Sizing Router Buffers". *Proc. 2004 ACM SIGCOMM* (Portland, ago. 2004).

ASH, G. R. *Dynamic Routing in Telecommunications Networks*. Nova York: McGraw Hill, 1998.

ASO-ICANN 2012. The Address Supporting Organization home page, <http://www.aso.icann.org>.

AT&T. "AT&T High Speed Internet Business Edition Service Level Agreements". <http://www.att.com/gen/general?pid=6622>.

ATHEROS COMMUNICATIONS INC. "Atheros AR5006 WLAN Chipset Product Bulletins". <http://www.atheros.com/pt/AR5006Bulletins.htm>.

AUGUSTIN, B.; KRISHNAMURTHY, B.; WILLINGER, W· "IXPs: Mapped?". *Proc. Internet Measurement Conference (IMC)*, nov. 2009.

AYANOGLU, E.; PAUL, S.; LA PORTA, T. F.; SABNANI, K. K.; GITLIN, R. D. "AIRMAIL: A Link-Layer Protocol for Wireless Networks". *ACM ACM/Baltzer Wireless Networks Journal*, 1: 47–60, fev. 1995.

BAKRE, A.; BADRINATH, B. R. "I-TCP: Indirect TCP for Mobile Hosts". *Proc. 1995 Int. Conf. on Distributed Computing Systems (ICDCS)* (maio 1995), p. 136–143.

BALAKRISHNAN, H.; PADMANABHAN, V.; SESHAN, S.; KATZ, R. "A Comparison of Mechanisms for Improving TCP Performance Over Wireless Links". *IEEE/ACM Transactions on Networking* v.5, n.6 (dez. 1997).

BALAKRISHNAN, H.; KAASHOEK, F.; KARGER, D.; MORRIS, R.; STOICA, I. "Looking Up Data in P2P Systems". *Communications of the ACM,* v.46, n.2 (fev. 2003), p. 43–48.

BALDAUF, M.; DUSTDAR, S.; ROSENBERG, F. "A Survey on Context-Aware Systems". *Int. J. Ad Hoc and Ubiquitous Computing*, v.2, n.4 (2007), p. 263–277.

BALLANI, H.; FRANCIS, P.; RATNASAMY, S. "A Measurement-based Deployment Proposal for IP Anycast". *Proc. 2006 ACM Internet Measurement Conf.*

BALLANI, H.; COSTA, P.; KARAGIANNIS, T.; ROWSTRON, A. "Towards Predictable Datacenter Networks". *Proc. 2011 ACM SIGCOMM.*

BARAN, P. "On Distributed Communication Networks". *IEEE Transactions on Communication Systems,* mar. 1964. Rand Corporation Technical report with the same title (Memorandum RM-3420-PR, 1964). <http://www.rand.org/publications/RM/RM3420/>.

BARDWELL, J. "You Believe You Understand What You Think I Said... The Truth About 802.11 Signal And Noise Metrics: A Discussion Clarifying Often-Misused 802.11 WLAN Terminologies". <http://www.connect802.com/download/techpubs/2004/you_believe_D100201.pdf>.

BARFORD, P.; DUFFIELD, N.; RON, A.; SOMMERS, J. "Network Performance Anomaly Detection and Localization". *Proc. 2009 IEEE INFOCOM* (abr. 2009).

BARONTI, P.; PILLAI, P.; CHOOK, V.; CHESSA, S.; GOTTA, A.; HU, Y. "Wireless Sensor Networks: A Survey on the State of the Art and the 802.15.4 and ZigBee Standards". *Computer Communications*, v.30, n.7 (2007), p. 1655–1695.

BASSET, S. A.; SCHULZRINNE, H. "An analysis of the Skype peer-to-peer Internet Telephony Protocol". *Proc. 2006 IEEE INFOCOM* (Barcelona, abr. 2006).

BBC NEWS ONLINE. "A Small Slice of Design". abr. 2001, <http://news.bbc.co.uk/2/hi/science/nature/1264205.stm>.

BBC. "Multicast". <http://www.bbc.co.uk/multicast/>.

BEHESHTI, N.; GANJALI, Y.; GHOBADI, M.; MCKEOWN, N.; SALMON, G. "Experimental Study of Router Buffer Sizing". *Proc. ACM Internet Measurement Conference* (Vouliagmeni, out. 2008).

BENDER, P.; BLACK, P.; GROB, M.; PADOVANI, R.; SINDHUSHAYANA, N.; VITERBI, A. "CDMA/HDR: A bandwidth-efficient high-speed wireless data service for nomadic users". *IEEE Commun.Mag.*, v.38, n.7 (jul. 2000) p. 70–77.

BERNERS-LEE, T.; CERN, "Information Management: A Proposal". mar. 1989, maio 1990. <http://www.w3.org/History/1989/proposal.html>.

BERNERS-LEE, T.; CAILLIAU, R.; LUOTONEN, A.; FRYSTYK NIELSEN, H.; SECRET, A. "The World-Wide Web". *Communications of the ACM,* v.37, n.8 (ago. 1994), p. 76–82.

BERTSEKAS, D.; GALLAGHER, R. *Data Networks,* 2.ed. Englewood Cliffs: Prentice Hall, 1991.

BIDDLE, P.; ENGLAND, P.; PEINADO, M.; WILLMAN, B. "The Darknet and the Future of Content Distribution". *2002 ACM Workshop on Digital Rights Management,* (Washington, nov. 2002,) <http://crypto.stanford.edu/DRM2002/darknet5.doc>.

BIERSACK, E. W. "Performance evaluation of forward error correction in ATM networks". *Proc. 1999 ACM SIGCOMM* (Baltimore, ago. 1992), p. 248–257.

BIND 2012. Internet Software Consortium page on BIND, <http://www.isc.org/bind.html>.

BISDIKIAN, C. "An Overview of the Bluetooth Wireless Technology". *IEEE Communications Magazine*, n.12 (dez. 2001), p. 86–94.

BISHOP, M. *Computer Security: Art and Science.* Boston: Addison Wesley, 2003.

BLACK, U. *ATM Volume I: Foundation for Broadband Networks.* Prentice Hall, 1995.

BLACK, U. *ATM Volume II: Signaling in Broadband Networks*. Prentice Hall, 1997.

BLUMENTHAL, M.; CLARK, D. "Rethinking the Design of the Internet: the End-to-end Arguments vs. the Brave New World". *ACM Transactions on Internet Technology*, v.1, n.1 (ago. 2001), p. 70–109.

BOCHMANN, G. V.; SUNSHINE, C. A. "Formal methods in communication protocol design". *IEEE Transactions on Communications*, v.28, n.4 (abr. 1980) p. 624–631.

BOLOT, J-C.; TURLETTI, T. "A rate control scheme for packet video in the Internet". *Proc. 1994 IEEE INFOCOM*, p. 1216–1223.

BOLOT, J-C.; VEGA-GARCIA, A. "Control Mechanisms for Packet Audio in the Internet". *Proc. 1996 IEEE INFOCOM*, p. 232–239.

BRADNER, S.; MANKIN, A. *IPng: Internet Protocol Next Generation*. Reading: Addison-Wesley, 1996.

BRAKMO, L.; PETERSON, L. "TCP Vegas: End to End Congestion Avoidance on a Global Internet". *IEEE Journal of Selected Areas in Communications*, v.13, n.8 (out. 1995), p. 1465–1480.

BRESLAU, L.; KNIGHTLY, E.; SHENKER, S.; STOICA, I.; ZHANG, H. "Endpoint Admission Control: Architectural Issues and Performance". *Proc. 2000 ACM SIGCOMM* (Estocolmo, ago. 2000).

BRYANT, B. "Designing an Authentication System: A Dialogue in Four Scenes". <http://web.mit.edu/kerberos/www/dialogue.html>.

BUSH, V. "As We May Think". *The Atlantic Monthly*, jul. 1945. <http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm>.

BYERS, J.; LUBY, M.; MITZENMACHER, M.; REGE, A. "A digital fountain approach to reliable distribution of bulk data". *Proc. 1998 ACM SIGCOMM* (Vancouver, ago. 1998), p. 56–67.

CABLELABS 2012. <http://www.cablelabs.com>.

CACHELOGIC 2012. <http://www.cachelogic.com>.

CAESAR, M.; CALDWELL, D.; FEAMSTER, N.; REXFORD, J.; SHAIKH, A.; VAN der MERWE, J. "Design and implementation of a Routing Control Platform". *Proc. Networked Systems Design and Implementation* (maio 2005a).

CAESAR, M.; REXFORD, J. "BGP Routing Policies in ISP Networks". *IEEE Network Magazine*, v.19, n.6 (nov. 2005b).

CALDWELL, C. "The Prime Pages". <http://www.utm.edu/research/primes/prove>.

CARDWELL, N.; SAVAGE, S.; ANDERSON, T. "Modeling TCP Latency". *Proc. 2000 IEEE INFOCOM* (Tel-Aviv, mar. 2000).

CASA 2012. Center for Collaborative Adaptive Sensing of the Atmosphere, <http://www.casa.umass.edu>.

CASADO, M.; FREEDMAN, M.; PETTIT, J.; LUO, J.; GUDE, N.; MCKEOWN, N.; SHENKER, S. "Rethinking Enterprise Network Control". *IEEE/ACM Transactions on Networking (ToN)*, v.17, n.4 (ago. 2009), p. 1270–1283.

CASADO, M.; FREEDMAN, M.; PETTIT, J.; LUO, J.; MCKEOWN, N.; SHENKER, S. "Ethane: Taking Control of the Enterprise". *Proc. 2007 ACM SIGCOMM* (Kioto, ago. 2007).

CASNER, S.; DEERING, S. "First IETF Internet Audiocast". *ACM SIGCOMM Computer Communications Review*, v.22, n.3 (jul. 1992), p. 92–97.

CEIVA 2012. <http://www.ceiva.com/>.

CENS 2012. Center for Embedded Network Sensing, <http://www.cens.ucla.edu/>.

CERF, V.; KAHN, R. "A Protocol for Packet Network Interconnection". *IEEE Transactions on Communications Technology*, v.COM-22, n.5, p. 627–641.

CERT 2001–09. "Advisory 2001–09: Statistical Weaknesses in TCP/IP Initial Sequence Numbers". <http://www.cert.org/advisories/CA-2001-09.html>.

CERT 2003–04. "CERT Advisory CA-2003-04 MS-SQL Server Worm". <http://www.cert.org/advisories/CA-2003-04.html>.

CERT 2012. <http://www.cert.org/advisories>.

CERT FILTERING 2012. “Packet Filtering for Firewall Systems”. <http://www.cert.org/tech_tips/packet_filtering.html>.

CERT SYN 1996. “Advisory CA-96.21: TCP SYN Flooding and IP Spoofing Attacks”. <http://www.cert.org/advisories/CA-1998-01.html>.

CHAO, H. J.; LAM, C.; OKI, E. *Broadband Packet Switching Technologies — A Practical Guide to ATM Switches and IP Routers*. John Wiley & Sons, 2001.

CHAO, H. J.; ZHANG, C.; DUNGHEL, P.; WU, D.; ROSS, K. W. “Unraveling the BitTorrent Ecosystem”. *IEEE Transactions on Parallel and Distributed Systems*, v.22, n.7 (jul. 2011).

CHEN, G.; KOTZ, D. “A Survey of Context-Aware Mobile Computing Research”. *Technical Report TR2000-381*, Dept. of Computer Science, Dartmouth College, nov. 2000. <http://www.cs.dartmouth.edu/reports/TR2000-381.pdf>.

CHEN, K.-T.; HUANG, C.-Y.; HUANG, P.; LEI, C.-L. “Quantifying Skype User Satisfaction”. *Proc. 2006 ACM SIGCOMM* (Pisa, Itália, set. 2006).

CHEN, K.; GUO, C.; WU, H.; YUAN, J.; FENG, Z.; CHEN, Y.; LU, S.; WU, W. “Generic and Automatic Address Configuration for Data Center Networks”. *Proc. 2010 ACM SIGCOMM*.

CHEN, Y.; JAIN, S.; ADHIKARI, V. K.; ZHANG, Z. “Characterizing Roles of Front-End Servers in End-to-End Performance of Dynamic Content Distribution”. *Proc. 2011 ACM Internet Measurement Conference* (Berlim, nov. 2011).

CHENOWETH, T.; MINCH, R.; TABOR, S. “Wireless Insecurity: Examining User Security Behavior on Public Networks”. *Communications of the ACM*, v.53, n.2 (fev. 2010), p. 134–138.

CHESWICK, B.; BURCH, H.; BRANIGAN, S. “Mapping and Visualizing the Internet”. *Proc. 2000 Usenix Conference* (San Diego, jun. 2000).

CHIU, D.; JAIN, R. “Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks”. *Computer Networks and ISDN Systems*, v.17, n.1, p. 1–14. <http://www.cs.wustl.edu/~jain/papers/cong_av.htm>.

CHRISTIANSEN, M.; JEFFAY, K.; OTT, D.; SMITH, F. D. “Tuning Red for Web Traffic”. *IEEE/ACM Transactions on Networking*, v.9, n.3 (jun. 2001), p. 249–264.

CHU, Y.; RAO, S.; SESHAN, S.; ZHANG, H. “A Case for End System Multicast”. *IEEE J. Selected Areas in Communications*, v.20, n.8 (out. 2002), p. 1456–1471.

CHUANG, S.; IYER, S.; MCKEOWN, N. “Practical Algorithms for Performance Guarantees in Buffered Crossbars”. *Proc. 2005 IEEE INFOCOM*.

CICCONETTI, C.; LENZINI, L.; MINGOZI, A.; EKLUND, K. “Quality of Service Support in 802.16 Networks”. *IEEE Network Magazine* (mar./abr. 2006), p. 50–55.

CISCO 12000 2012. “Cisco XR 12000 Series and Cisco 12000 Series Routers”. <http://www.cisco.com/en/US/products/ps6342/index.html>.

CISCO 8500 2012. “Catalyst 8500 Campus Switch Router Architecture”. <http://www.cisco.com/univercd/cc/td/doc/product/l3sw/8540/rel_12_0/w5_6f/softcnfg/1cfg8500.pdf>.

CISCO 2011. *Cisco Visual Networking Index: Forecast and Methodology, 2010–2015*. White Paper, 2011.

CISCO 2012. http://www.cisco.com/go/dce.

CISCO NAT 2012. “How NAT Works”. <http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080094831.shtml>.

CISCO QoS 2012. “Advanced QoS Services for the Intelligent Internet”. <http://www.cisco.com/warp/public/cc/pd/iosw/ioft/ioqo/tech/qos_wp.htm>.

CISCO QUEUE 2012. “Congestion Management Overview”. <http://www.cisco.com/en/US/docs/ios/12_2/qos/configuration/guide/qcfconmg.html>.

CISCO SWITCHES 2012. “Multiservice Switches”. <http://www.cisco.com/warp/public/cc/pd/si/index.shtml>.

CISCO SYN 2012. “Defining Strategies to Protect Against TCP SYN Denial of Service Attacks”. <http://www.cisco.com/en/US/tech/tk828/technologies_tech_ note09186a00800f67d5.shtml>.

CISCO VNI 2011. "Visual Networking Index". <http://www.cisco.com/web/solutions/sp/vni/vni_forecast_highlights/index.html>.

CLARK, D. "The Design Philosophy of the DARPA Internet Protocols". *Proc. 1988 ACM SIGCOMM* (Stanford, ago. 1988).

CLARKE, I.; HONG, T. W.; MILLER, S. G.; SANDBERG, O.; WILEY, B. "Protecting Free Expression Online with Freenet". *IEEE Internet Computing* (jan.–fev. 2002), p. 40–49.

COHEN, D. "Issues in Transnet Packetized Voice Communication". *Proc. Fifth Data Communications Symposium* (Snowbird, set. 1977), p. 6–13.

COHEN, B. "Incentives to Build Robustness in BitTorrent". *First Workshop on the Economics of Peer-to-Peer Systems* (Berkeley, jun.2003).

COOKIE CENTRAL 2012. <http://www.cookiecentral.com/n_cookie_faq.htm>.

CORMEN, T. H. *Introduction to Algorithms*. 2.ed., Cambridge: MIT Press, 2001.

CROW, B.; WIDJAJA, I.; KIM, J.; SAKAI, P. "IEEE 802.11 Wireless Local Area Networks". *IEEE Communications Magazine* (set. 1997), p. 116–126.

CROWCROFT, J.; WANG, Z.; SMITH, A.; ADAMS, J. "A Comparison of the IETF and ATM Service Models". *IEEE Communications Magazine* (nov./dez. 1995), p. 12–16.

CROWCROFT, J.; HANDLEY, M.; WAKEMAN, I. *Internetworking Multimedia*. San Francisco: Morgan-Kaufman, 1999.

CURTIS, A. R.; MOGUL, J. C.; TOURRILHES, J.; YALAGANDULA, P.; SHARMA, P.; BANERJEE, S. "DevoFlow: Scaling Flow Management for High-Performance Networks". *Proc. 2011 ACM SIGCOMM*.

CUSUMANO, M. A.; YOFFIE, D. B. *Competing on Internet Time: Lessons from Netscape and its Battle with Microsoft*. Nova York: Free Press, 1998.

DAHLMAN, E.; GUDMUNDSON, B.; NILSSON, M.; SKÖLD, J. "UMTS/IMT-2000 Based on Wideband CDMA". *IEEE Communications Magazine* (set. 1998), p. 70–80.

DAIGLE, J. N. *Queuing Theory for Telecommunications*. Reading: Addison-Wesley, 1991.

DALAL, Y.; METCALFE R., "Reverse Path Forwarding of Broadcast Packets". *Communications of the ACM*, v.21, n.12 (dez. 1978), p. 1040–1048.

DAVIE B.; REKHTER, Y. *MPLS: Technology and Applications*. Morgan Kaufmann Series in Networking, 2000.

DAVIES, G.; KELLY, F. "Network Dimensioning, Service Costing, and Pricing in a Packet-Switched Environment". *Telecommunications Policy*, v.28, n.4, p. 391–412.

DEC 1990. "In Memoriam: J. C. R. Licklider 1915–1990". SRC Research Report 61, ago. 1990. <http://www.memex.org/licklider.pdf>.

DECLERCQ, J.; PARIDAENS, O. "Scalability Implications of Virtual Private Networks". *IEEE Communications Magazine*, v.40, n.5 (maio 2002), p. 151–157.

DEMERS, A.; KESHAV, S.; SHENKER, S. "Analysis and Simulation of a Fair Queuing Algorithm". *Internetworking: Research and Experience*, v.1, n.1 (1990), p. 3–26.

DENNING, D. (Ed.), DENNING, P. (Pref.). *Internet Besieged: Countering Cyberspace Scofflaws*. Reading: Addison-Wesley, 1997.

DHC 2012 IETF. Dynamic Host Configuration working group homepage. <http://www.ietf. org/html.charters/dhc-charter.html>.

DHUNGEL, P.; ROSS, K. W.; STEINER., M.; TIAN, Y.; HEI, X. "Xunlei: Peer-Assisted Download Acceleration on a Massive Scale". *Passive and Active Measurement Conference (PAM) 2012*, Viena, 2012.

DHUNGEL, P.; WU, D.; SCHONHORST, B.; ROSS, K. W. "A Measurement Study of Attacks on BitTorrent Leechers". *7th International Workshop on Peer-to-Peer Systems (IPTPS 2008)* (Tampa Bay, fev. 2008).

DIFFIE, W.; HELLMAN, M. E. "New Directions in Cryptography". *IEEE Transactions on Information Theory*, v.IT-22 (1976), p. 644–654.

DIGGAVI, S. N.; AL-DHAHIR, N.; STAMOULIS, A.; CALDERBANK, R. "Great Expectations: The Value of Spatial Diversity in Wireless Networks". *Proceedings of the IEEE*, v.92, n.2 (fev. 2004).

DILLEY, J.; MAGGS, B.; PARIKH, J.; PROKOP, H.; SITARAMAN, R.; WEIHL, B. "Globally Distributed Content Delivert". *IEEE Internet Computing* (set.-out. 2002).

DING, Y.; DU, Y.; HU, Y.; LIU, Z.; WANG, L.; ROSS, K. W.; GHOSE, A. "Broadcast Yourself: Understanding YouTube Uploaders". *Proc. 2011 ACM Internet Measurement Conference* (Berlim).

DIOT, C.; LEVINE, B. N.; LYLES, B.; KASSEM, H.; BALENSIEFEN, D. "Deployment Issues for the IP Multicast Service and Architecture". *IEEE Network*, v.14, n.1 (jan./fev. 2000) p. 78–88.

DISCHINGER, M.; HAEBERLEN, A.; GUMMADI, K.; SAROIU, S. "Characterizing residential broadband networks". *Proc. 2007 ACM Internet Measurement Conference*, p. 24–26.

DMITIROPOULOS, X.; KRIOUKOV, D.; FOMENKOV, M.; HUFFAKER, B.; HYUN, Y.; CLAFFY, K. C.; RILEY, G. "AS Relationships: Inference and Validation". *ACM Computer Communication Review* (jan. 2007).

DOCSIS 2004. Data-over-cable service interface specifications: Radio-frequency interface specification.ITU-T J.112, 2004.

DOCSIS 2011. Data-Over-Cable Service Interface Specifications, DOCSIS 3.0: MAC and Upper Layer Protocols Interface Specification, CM-SP-MULPIv3.0-I16-110623, 2011.

DODGE, M. "An Atlas of Cyberspaces". <http://www.cybergeography.org/atlas/isp_maps.html>.

DONAHOO, M.; CALVERT, K. *TCP/IP Sockets in C: Practical Guide for Programmers*. Morgan Kaufman, 2001.

DOUCEUR, J. R. "The Sybil Attack". *First International Workshop on Peer-to-Peer Systems* (*IPTPS '02*) (Cambridge, mar. 2002).

DSL 2012. DSL Forum homepage, <http://www.dslforum.org/>.

DROMS, R.; LEMON, T. *The DHCP Handbook*. 2.ed. SAMS Publishing, 2002.

EDNEY, J.; ARBAUGH, W. A. *Real 802.11 Security: Wi-Fi Protected Access and 802.11i*. Addison-Wesley Professional, 2003.

EDWARDS, W. K.; GRINTER, R.; MAHAJAN, R.; WETHERALL, D. "Advancing the State of Home Networking". *Communications of the ACM*, v.54, n.6 (jun. 2011), p. 62–71.

EKLUND, K.; MARKS, R.; STANSWOOD, K.; WANG, S. "IEEE Standard 802.16: A Technical Overview of the Wireless MAN Air Interface for Broadband Wireless Access". *IEEE Communications Magazine* (jun. 2002), p. 98–107.

ELLIS, H. "The Story of Non-Secret Encryption". http://jya.com/ellisdoc.htm.

ERICSSON 2011. "LTE—An Introduction". <www.ericsson.com/res/docs/2011/lte_an_introduction.pdf>.

ERICSSON 2012. "The Evolution of Edge". <http://www.ericsson.com/technology/whitepapers/broadband/evolution_of_EDGE.shtml>.

ESTRIN, D.; HANDLEY, M.; HELMY, A.; HUANG, P.; THALER, D. "A Dynamic Bootstrap Mechanism for Rendezvous-Based Multicast Routing". *Proc. 1998 IEEE INFOCOM* (Nova York, abr. 1998).

FALKNER, J.; PIATEK, M.; JOHN, J. P.; KRISHNAMURTHY, A.; ANDERSON, T. "Profiling a Million Sser DHT". *Proc. 2007 ACM Internet Measurement Conference.*

FALOUTSOS, C.; FALOUTSOS, M.; FALOUTSOS, P. "What Does the Internet Look Like? Empirical Laws of the Internet Topology". *Proc. 1999 ACM SIGCOMM* (Boston, ago. 1999).

FARRINGTON, N.; PORTER, G.; RADHAKRISHNAN, S.; BAZZAZ, H.; SUBRAMANYA, V.; FAINMAN, Y.; PAPEN, G.; VAHDAT, A. "Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers". *Proc. 2010 ACM SIGCOMM.*

FEAMSTER, N.; WINICK, J.; REXFORD, J. "A Model for BGP Routing for Network Engineering". *Proc. 2004 ACM SIGMETRICS* (Nova York, jun. 2004).

FEAMSTER, N.; BALAKRISHNAN, H. "Detecting BGP Configuration Faults with Static Analysis". *NSDI* (maio 2005).

FELDMAN, M.; CHUANG, J. "Overcoming Free-Riding Behavior in Peer-to-peer Systems". *ACM SIGecom Exchanges* (jul. 2005).

FELDMEIER, D. "Fast Software Implementation of Error Detection Codes". *IEEE/ACM Transactions on Networking*, v.3, n.6 (dez. 1995), p. 640–652.

FIPS 1995. Federal Information Processing Standard, "Secure Hash Standard". FIPS Publication 180-1. <http://www.itl.nist.gov/fipspubs/fip180-1.htm>.

FLOYD, S.; FALL, K. "Promoting the Use of End-to-End Congestion Control in the Internet". *IEEE/ACM Transactions on Networking*, v.6, n.5 (out. 1998), p. 458–472.

FLOYD, S.; HANDLEY, M.; PADHYE, J.; WIDMER, J. "Equation-Based Congestion Control for Unicast Applications". *Proc. 2000 ACM SIGCOMM* (Estocolmo, ago. 2000).

FLOYD, S. "A Report on Some Recent Developments in TCP Congestion Control". *IEEE Communications Magazine* (abr. 2001).

FLOYD, S. "References on RED (Random Early Detection) Queue Management". <http://www.icir.org/floyd/red.html>.

FLOYD, S.; JACOBSON, V. "Synchronization of Periodic Routing Messages". *IEEE/ACM Transactions on Networking*, v.2, n.2 (abr. 1997) p. 122–136.

FLOYD, S. "TCP and Explicit Congestion Notification". *ACM SIGCOMM Computer Communications Review*, v.24, n.5 (out. 1994), p. 10–23.

FLUHRER, S.; MANTIN, I.; SHAMIR, A. "Weaknesses in the Key Scheduling Algorithm of RC4". *Eighth Annual Workshop on Selected Areas in Cryptography*, (Toronto, ago. 2002).

FORTZ, B.; THORUP, M. "Internet Traffic Engineering by Optimizing OSPF Weights". *Proc. 2000 IEEE INFOCOM* (Tel Aviv, abr. 2000).

FORTZ, B.; REXFORD, J.; THORUP, M. "Traffic Engineering with Traditional IP Routing Protocols". *IEEE Communication Magazine* (out. 2002).

FRALEIGH, C.; TOBAGI, F.; DIOT, C. "Provisioning IP Backbone Networks to Support Latency Sensitive Traffic". *Proc. 2003 IEEE INFOCOM* (San Francisco, mar. 2003).

FREEDMAN, M. J.; FREUDENTHAL, E.; MAZIRES, D. "Democratizing Content Publication with Coral". *USENIX NSDI*, 2004.

FRIEDMAN, T.; TOWSLEY, D. "Multicast Session Membership Size Estimation". *Proc. 1999 IEEE INFOCOM* (Nova York, mar. 1999).

FROST, J. "BSD Sockets: A Quick and Dirty Primer". <http://world.std.com/~jimf/papers/sockets/sockets.html>.

FTTH COUNCIL 2011a. "NORTH AMERICAN FTTH STATUS—MARCH 31, 2011" (mar. 2011). <www.ftthcouncil.org>.

FTTH COUNCIL 2011b. "2011 Broadband Consumer Research" (jun. 2011), <www.ftthcouncil.org>.

GALLAGHER, R. G.; HUMBLET, P. A.; SPIRA, P. M. "A Distributed Algorithm for Minimum Weight-Spanning Trees". *ACM Trans. on Programming Languages and Systems*, v.1, n.5 (jan. 1983), p. 66–77.

GAO, L.; REXFORD, J. "Stable Internet Routing Without Global Coordination". *IEEE/ACM Transactions on Networking*, v.9, n.6 (dez. 2001), p. 681–692.

GARCES-ERCE, L.; ROSS, K. W.; BIERSACK, E.; FELBER, P.; URVOY-KELLER, G. "TOPLUS: Topology Centric Lookup Service". *Fifth Int. Workshop on Networked Group Communications* (*NGC 2003*) (Munique, set. 2003) <http://cis.poly.edu/~ross/papers/TOPLUS.pdf>.

GARTNER, F. C. "A Survey of Self-Stabilizing Spanning-Tree Construction Algorithms". *Technical Report IC/2003/38*, Swiss Federal Institute of Technology (EPFL), School of Computer and Communication Sciences, jun.10, 2003. <http://ic2.epfl.ch/publications/documents/IC_TECH_REPORT_200338.pdf>.

GAUTHIER, L.; DIOT, C.; KUROSE, J. "End-to-end Transmission Control Mechanisms for Multiparty Interactive Applications on the Internet". *Proc. 1999 IEEE INFOCOM* (Nova York, abr. 1999).

GIRARD, A. *Routing and Dimensioning in Circuit-Switched Networks*. Reading: Addison-Wesley, 1990.

GLITHO, R. "Contrasting OSI Systems Management to SNMP and TMN". *Journal of Network and Systems Management,* v.6, n.2 (jun. 1998), p. 113–131.

GNUTELLA 2009. "The Gnutella Protocol Specification, v0.4" <http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf>.

GOODMAN, D. J. *Wireless Personal Communications Systems.* Prentice-Hall, 1997.

GOOGLE LOCATIONS 2012. Google data centers. <http://www.google.com/corporate/datacenter/locations.html>.

GORALSKI, W. *Frame Relay for High-Speed Networks.* Nova York: John Wiley, 1999.

GORALSKI, W. *Optical Networking and WDM.* Berkeley: Osborne/McGraw-Hill, 2001.

GREENBERG, A.; HAMILTON, J.; MALTZ, D.; PATEL, P. "The Cost of a Cloud: Research Problems in Data Center Networks". *ACM Computer Communications Review* (jan. 2009a).

GREENBERG, A.; JAIN, N.; KANDULA, S.; KIM, C.; LAHIRI, P.; MALTZ, D.; PATEL, P.; SENGUPTA, S. "VL2: A Scalable and Flexible Data Center Network". *Proc. 2009 ACM SIGCOMM.* (2009b)

GREENBERG, A.; HAMILTON, J.; JAIN, N.; KANDULA, S.; KIM, C.; LAHIRI, P.; MALTZ, D.; PATEL, P.; SENGUPTA, S. "VL2: A Scalable and Flexible Data Center Network". *Communications of the ACM,* v.54, n.3 (mar. 2011), p. 95–104.

GRIFFIN, T. "Interdomain Routing Links". <http://www.cl.cam.ac.uk/~tgg22/interdomain/>.

GUHA, S.; DASWANI, N.; JAIN, R. "An Experimental Study of the Skype Peer-to-Peer VoIP System". *Proc. Fifth Int. Workshop on P2P Systems* (Santa Barbara, 2006).

GUO, L.; CHEN, S.; XIAO, Z.; TAN, E.; DING, X.; ZHANG, X. "Measurement, Analysis, and Modeling of BitTorrent-Like Systems". *Proc. 2005 ACM Internet Measurement Conference.*

GUO, C.; LU, G.; LI, D.; WU, H.; ZHANG, X.; SHI, Y.; TIAN, C.; ZHANG, Y.; LU, S. "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers". *Proc. 2009 ACM SIGCOMM.*

GUPTA, P.; MCKEOWN, N. "Algorithms for Packet Classification". *IEEE Network Magazine,* v.15, n.2 (mar./abr. 2001), p. 24–32.

HA, S., RHEE, I., XU, L. "CUBIC: A New TCP-Friendly High-Speed TCP Variant". *ACM SIGOPS Operating System Review,* 2008.

HALABI, S. *Internet Routing Architectures.* 2.ed., Cisco Press, 2000.

HALPERIN, D.; HEYDT-BENJAMIN, T.; RANSFORD, B.; CLARK, S.; DEFEND, B.; MORGAN, W.; FU, K.; KOHNO, T.; MAISEL, W. "Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses". *Proc. 29th Annual IEEE Symposium on Security and Privacy* (maio 2008).

HALPERIN, D.; KANDULA, S.; PADHYE, J.; BAHL, P.; WETHERALL, D. "Augmenting Data Center Networks with Multi-Gigabit Wireless Links". *Proc. 2011 ACM SIGCOMM.*

HANBALI, A. A.; ALTMAN, E.; NAIN, P. "A Survey of TCP over Ad Hoc Networks". *IEEE Commun.Surveys and Tutorials,* v.7, n.3 (2005), p. 22–36.

HEI, X.; LIANG, C.; LIANG, J.; LIU, Y.; ROSS, K. W. "A Measurement Study of a Large-scale P2P IPTV System". *IEEE Trans. on Multimedia* (dez. 2007).

HEIDEMANN, J.; OBRACZKA, K.; TOUCH, J. "Modeling the Performance of HTTP over Several Transport Protocols". *IEEE/ACM Transactions on Networking,* v.5, n.5 (out. 1997), p. 616–630.

HELD, G. *Data Over Wireless Networks: Bluetooth, WAP, and Wireless LANs.* McGraw-Hill, 2001.

HERSENT, O.; GURLE, D.; PETIT, J-P. *IP Telephony: Packet-Based Multimedia Communication Systems.* Edinburgh: Pearson Education Ltd., 2000.

HOLLAND, G.; VAIDYA, N.; BAHL, V. "A Rate-Adaptive MAC Protocol for Multi-Hop Wireless Networks". *Proc. 2001 ACM Int. Conference of Mobile Computing and Networking (Mobicom01)* (Roma, jul. 2001).

HOLLOT, C. V.; MISRA, V.; TOWSLEY, D.; GONG, W. "Analysis and design of controllers for AQM routers supporting TCP flows". *IEEE Transactions on Automatic Control,* v.47, n.6 (jun. 2002), p. 945–959.

HUANG, C.; SHARMA, V.; OWENS, K.; MAKAM, V. "Building Reliable MPLS Networks Using a Path Protection Mechanism". *IEEE Communications Magazine,* v.40, n.3 (mar. 2002), p. 156–162.

HUANG, Y.; GUERIN, R. "Does Over-Provisioning Become More or Less Efficient as Networks Grow Larger?". *Proc. IEEE Int. Conf. Network Protocols* (*ICNP*) (Boston, nov. 2005).

HUANG, C.; LI, J.; ROSS, K. W. "Can Internet VoD Be Profitable?". *Proc 2007 ACM SIGCOMM* (Kioto, ago. 2007).

HUANG, C.; LI, J.; WANG, A.; ROSS, K. W. "Understanding Hybrid CDN-P2P: Why Limelight Needs its Own Red Swoosh". *Proc. 2008 NOSSDAV*, Braunschweig.

HUANG, C.; HOLT, N.; WANG, Y. A.; GREENBERG, A.; LI, J.; ROSS, K. W. "A DNS Reflection Method for Global Traffic Management". *Proc. 2010 USENIX,* Boston.

HUITEMA, C. *IPv6: The New Internet Protocol.* 2.ed. Englewood Cliffs: Prentice Hall, 1998.

HUSTON, G. "Interconnection, Peering, and Settlements — Part I". *The Internet Protocol Journal,* v.2, n.1 (mar. 1999a).

HUSTON, G. "NAT Anatomy: A Look Inside Network Address Translators". *The Internet Protocol Journal,* v.7, n.3 (set. 2004).

HUSTON, G. "Confronting IPv4 Address Exhaustion". <http://www.potaroo.net/ispcol/2008-10/v4depletion.html>.

HUSTON, G.; MICHAELSON, G. "IPv6 Deployment: Just where are we?". <http://www.potaroo.net/ispcol/2008-04/ipv6.html>.

HUSTON, G. "A Rough Guide to Address Exhaustion". *The Internet Protocol Journal,* v.14, n.1 (mar. 2011).

HUSTON, G. "Transitioning Protocols". *The Internet Protocol Journal,* v.14, n.1 (mar. 2011).

IAB 2012. <http://www.iab.org/>.

IANA 2012a. <http://www.iana.org/>.

IANA 2012b. "Private Enterprise Numbers". <http://www.iana.org/assignments/enterprise-numbers>.

IANA PROTOCOL NUMBERS 2012. <http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>.

IANA TLD 2012. <http://www.iana.org/domains/root/db/>.

ICANN 2012. <http://www.icann.org>.

IEC OPTICAL 2012. "Optical Access". <http://www.iec.org/online/tutorials/opt_acc/>.

IEEE 802 2012. <http://www.ieee802.org/>.

IEEE 802.11 1999. "1999 Edition (ISO/IEC 8802-11: 1999) IEEE Standards for Information Technology — Telecommunications and Information Exchange Between Systems — Local and Metropolitan Area Network — Specific Requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification". <http://standards.ieee.org/getieee802/download/802.11-1999.pdf>.

IEEE 802.11n 2012. "IEEE P802.11 — Task Group N — Meeting Update: Status of 802.11n". <http://grouper.ieee.org/groups/802/11/Reports/tgn_update.htm>.

IEEE 802.15 2012. IEEE 802.15 Working Group for WPAN homepage. <http://grouper.ieee. org/groups/802/15/>.

IEEE 802.15.4 2012. IEEE 802.15 WPAN Task Group 4. <http://www.ieee802.org/15/pub/TG4.html>.

IEEE 802.16d 2004. "IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems". <http://standards.ieee.org/getieee802/download/802.16-2004.pdf>.

IEEE 802.16e 2005. "IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems, Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1". <http://standards.ieee.org/getieee802/download/802.16e-2005.pdf>.

IEEE 802.1q 2005. "IEEE Standard for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks". <http://standards.ieee.org/getieee802/download/802.1Q-2005.pdf>.

IEEE 802.1X. IEEE Std 802.1X-2001 Port-Based Network Access Control, <http://standards.ieee.org/reading/ieee/std_public/description/lanman/802.1x-2001_desc.html>.

IEEE 802.3 2012. "IEEE 802.3 CSMA/CD (Ethernet)". <http://grouper.ieee.org/groups/802/3/>.

IEEE 802.5 2012. <http://www.ieee802.org/5/www8025org/>.

IETF 2012. <http://www.ietf.org>.

IHM, S.; PAI, V. S. "Towards Understanding Modern Web Traffic". *Proc. 2011 ACM Internet Measurement Conference* (Berlim).

IMAP 2012. The IMAP Connection. <http://www.imap.org/>.

INTEL 2012. "Intel® 82544 Gigabit Ethernet Controller". <http://www.intel.com/design/network/products/lan/docs/82544_docs.htm>.

INTEL WIMAX 2012. "WiMax Technology". <http://www.intel.com/technology/wimax/index.htm>.

INTERNET2 MULTICAST 2012. <http://www.internet2.edu/multicast/>.

IPv6 2012. <http://www.ipv6.com/>.

ISC 2012. <http://www.isc.org>.

ISI 1979. "DoD Standard Internet Protocol". Internet Engineering Note 123 (dez. 1979). <http://www.isi.edu/in-notes/ien/ien123.txt>.

ISO 2012. <http://www.iso.org/>.

ISO X.680 2002. "X.680: ITU-T Recommendation X.680 (2002) Information Technology — Abstract Syntax Notation One (ASN.1): Specification of Basic Notation". <http://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf>.

ITU 1999. Asymmetric Digital Subscriber Line (ADSL) Transceivers. ITU-T G.992.1, 1999.

ITU 2003. Asymmetric Digital Subscriber Line (ADSL) Transceivers — Extended Bandwidth ADSL2 (ADSL2Plus). ITU-T G.992.5, 2003.

ITU 2005a. "ITU-T X.509, The Directory: Public-key and attribute certificate frameworks" (ago. 2005).

ITU 2005b. *The Internet of Things*. 2005. <http://www.itu.int/osg/spu/publications/internetofthings/InternetofThings_summary.pdf>.

ITU 2012. <http://www.itu.int/>.

ITU Statistics 2012. International Telecommunications Union, "ICT Statistics". <http://www.itu.int/ITU-D/icteye/Reports.aspx>.

ITU 2011. ITU, "Measuring the Information Society, 2011". <http://www.itu.int/ITU-D/ict/publications/idi/2011/index.html>.

ITU 2011. "The World in 2010: ICT Facts and Figures". <http://www.itu.int/ITU-D/ict/material/Telecom09_flyer.pdf>.

ITU-T Q.2931 1995. "Recommendation Q.2931 (02/95) - Broadband Integrated Services Digital Network (B-ISDN) — Digital subscriber signalling system no. 2 (DSS 2) — User-network interface (UNI) — Layer 3 specification for basic call/connection control".

IYER, S.; ZHANG, R.; MCKEOWN, N. "Routers with a Single Stage of Buffering". *Proc. 2002 ACM SIGCOMM* (Pittsburgh, ago. 2002).

IYER, S.; KOMPELLA, R. R.; MCKEOWN, N. "Designing Packet Buffers for Router Line Cards". *IEEE Transactions on Networking*, v.16, n.3 (jun. 2008), p. 705–717.

JACOBSON, V. "Congestion Avoidance and Control". *Proc. 1988 ACM SIGCOMM* (Stanford, ago. 1988), p. 314–329.

JAIN, R. "A timeout-based congestion control scheme for window flow-controlled networks". *IEEE Journal on Selected Areas in Communications SAC-4*, 7 (out. 1986).

JAIN, R. "A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks". *ACM SIGCOMM Computer Communications Review*, v.19, n.5 (1989), p. 56–71.

JAIN, R. *FDDI Handbook: High-Speed Networking Using Fiber and Other Media*. Reading: Addison-Wesley, 1994.

JAIN, R.; KALYANARAMAN, S.; FAHMY, S.; GOYAL, R.; KIM, S. "Tutorial Paper on ABR Source Behavior". *ATM Forum*/96-1270, out. 1996. <http://www.cse.wustl.edu/~jain/atmf/ftp/atm96-1270.pdf>.

JAISWAL, S.; IANNACCONE, G.; DIOT, C.; KUROSE, J.; TOWSLEY, D. "Measurement and Classification of Out-of-Sequence Packets in a Tier-1 IP backbone". *Proc. 2003 IEEE INFOCOM.*

JI, P.; GE, Z.; KUROSE, J.; TOWSLEY, D. "A Comparison of Hard-State and Soft-State Signaling Protocols". *Proc. 2003 ACM SIGCOMM* (Karlsruhe, ago. 2003).

JIANG, W.; LENNOX, J.; SCHULZRINNE, H.; SINGH, K. "Towards Junking the PBX: Deploying IP Telephony". *NOSSDAV'01* (Port Jefferson, jun. 2001).

JIMENEZ, D. "Outside Hackers Infiltrate MIT Network, Compromise Security". *The Tech*, v.117, n.49 (out. 1997), p. 1, <http://www-tech.mit.edu/V117/N49/hackers.49n.html>.

JIN, C.; WE, D. X.; LOW, S. "FAST TCP: Motivation, architecture, algorithms, performance". *Proc. 2004 IEEE INFOCOM* (Hong Kong, mar. 2004).

KAARANEN, H.; NAGHIAN, S.; LAITINEN, L.; AHTIAINEN, A.; NIEMI, V. *Networks: Architecture, Mobility and Services.* Nova York: John Wiley & Sons, 2001.

KAHN, D. *The Codebreakers: The Story of Secret Writing.* The Macmillan Company, 1967.

KAHN, R. E.; GRONEMEYER, S.; BURCHFIEL, J.; KUNZELMAN, R. "Advances in Packet Radio Technology". *Proc. 1978 IEEE INFOCOM*, 66, 11 (nov. 1978).

KAMERMAN, A.; MONTEBAN, L.; "WaveLAN-II: A High–Performance Wireless LAN for the Unlicensed Band". *Bell Labs Technical Journal* (Verão 1997), p. 118–133.

KANGASHARJU, J.; ROSS, K. W.; ROBERTS, J. W. "Performance Evaluation of Redirection Schemes in Content Distribution Networks". *Proc. 5th Web Caching and Content Distribution Workshop* (Lisboa, maio 2000).

KAR, K.; KODIALAM, M.; LAKSHMAN, T. V. "Minimum Interference Routing of Bandwidth Guaranteed Tunnels with MPLS Traffic Engineering Applications". *IEEE J. Selected Areas in Communications* (dez. 2000).

KARN, P.; PARTRIDGE, C. "Improving Round-Trip Time Estimates in Reliable Transport Protocols". *Proc. 1987 ACM SIGCOMM.*

KAROL, M.; HLUCHYJ, M.; MORGAN, A. "Input Versus Output Queuing on a Space-Division Packet Switch". *IEEE Transactions on Communications,* v.35, n.12 (dez. 1987), p. 1347–1356.

KATABI, D.; HANDLEY, M.; ROHRS, C. "Internet Congestion Control for Future High Bandwidth-Delay Product Environments". *Proc. 2002 ACM SIGCOMM* (Pittsburgh, ago. 2002).

KATZELA, I.; SCHWARTZ. M.; "Schemes for Fault Identification in Communication Networks". *IEEE/ACM Transactions on Networking,* v.3, n.6 (dez. 1995), p. 753–764.

KAUFMAN, C.; PERLMAN, R.; SPECINER, M. *Network Security, Private Communication in a Public World.* Englewood Cliffs: Prentice Hall, 1995.

KELLY, F. P.; MAULLOO, A.; TAN, D. "Rate control for communication networks: Shadow prices, proportional fairness and stability". *J. Operations Res. Soc.*, v.49, n.3 (mar. 1998), p. 237–252.

KELLY, T. "Scalable TCP: improving performance in high speed wide area networks". *ACM SIGCOMM Computer Communications Review,* v.33, n.2 (abr. 2003), pp 83–91.

KILKKI, K. *Differentiated Services for the Internet.* Indianapolis: Macmillan Technical Publishing, 1999.

KIM, H.; RIXNER, S.; PAI, V. "Network Interface Data Caching". *IEEE Transactions on Computers*, v.54, n.11 (nov. 2005), p. 1394–1408.

KIM, C.; CAESAR, M.; REXFORD, J. "Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises". *Proc. 2008 ACM SIGCOMM* (Seattle, ago. 2008).

KLEINROCK, L. "Information Flow in Large Communication Networks". *RLE Quarterly Progress Report*, jul. 1961.

KLEINROCK, L. *1964 Communication Nets: Stochastic Message Flow and Delay.* Nova York: McGraw-Hill, 1964.

KLEINROCK, L. *Queuing Systems, v.1.* Nova York: John Wiley, 1975.

KLEINROCK, L.; TOBAGI, F. A. “Packet Switching in Radio Channels: Part I — Carrier Sense Multiple-Access Modes and Their Throughput-Delay Characteristics”. *IEEE Transactions on Communications,* v.23, n.12 (dez. 1975), p. 1400–1416.

KLEINROCK, L. *Queuing Systems, v.2.* Nova York: John Wiley, 1976.

KLEINROCK, L. “The Birth of the Internet”. <http://www.lk.cs.ucla.edu/LK/Inet/birth.html>.

KOHLER, E.; HANDLEY, M.; FLOYD, S. “DDCP: Designing DCCP: Congestion Control Without Reliability”. *Proc. 2006 ACM SIGCOMM* (Pisa, set. 2006).

KOLDING, T.; PEDERSEN, K.; WIGARD, J.; FREDERIKSEN, F.; MOGENSEN, P. “High Speed Downlink Packet Access: WCDMA Evolution”. *IEEE Vehicular Technology Society News* (fev. 2003), p. 4–10.

KOPONEN, T.; SHENKER, S.; BALAKRISHNAN, H.; FEAMSTER, N.; GANICHEV, I.; GHODSI, A.; GODFREY, P. B.; MCKEOWN, N.; PARULKAR, G.; RAGHAVAN, B.; REXFORD, J.; ARIANFAR, S.; KUPTSOV, D. “Architecting for Innovation”. *ACM Computer Communications Review*, 2011.

KORHONEN, J. *Introduction to 3G Mobile Communications.* 2.ed. Artech House, 2003.

KOZIOL, J. *Intrusion Detection with Snort.* Sams Publishing, 2003.

KRISHNAMURTHY, B.; REXFORD, J. *Web Protocols and Practice: HTTP/1.1, Networking Protocols, and Traffic Measurement.* Boston: Addison-Wesley, 2001a.

KRISHNAMURTHY, B.; WILLS, C.; ZHANG, Y. “On the Use and Performance of Content Distribution Networks”. *Proc. 2001 ACM Internet Measurement Conference.* (2001b)

KRISHNAN, R.; MADHYASTHA, H.; SRINIVASAN, S.; JAIN, S.; KRISHNAMURTHY, A.; ANDERSON, T.; GAO, J. “Moving Beyond End-to-end Path Information to Optimize CDN Performance”. *Proc. 2009 ACM Internet Measurement Conference.*

KULKARNI, S.; ROSENBERG, C. “Opportunistic Scheduling: Generalizations to Include Multiple Constraints, Multiple Interfaces, and Short Term Fairness”. *Wireless Networks*, 11 (2005), 557–569.

KUMAR, R.; ROSS, K.W. “Optimal Peer-Assisted File Distribution: Single and Multi-Class Problems”. *IEEE Workshop on Hot Topics in Web Systems and Technologies* (Boston, 2006).

LABOVITZ, C.; MALAN, G. R.; JAHANIAN, F. “Internet Routing Instability”. *Proc. 1997 ACM SIGCOMM* (Cannes, set. 1997), p. 115–126.

LABOVITZ, C.; IEKEL-JOHNSON, S.; MCPHERSON, D.; OBERHEIDE, J.; JAHANIAN, F. “Internet Inter-Domain Traffic”. *Proc. 2010 ACM SIGCOMM.*

LABRADOR, M.; BANERJEE, S. “Packet Dropping Policies for ATM and IP Networks”. *IEEE Communications Surveys*, v.2, n.3 (Third Quarter 1999), p. 2–14.

LACAGE, M.; MANSHAEI, M. H.; TURLETTI, T. “IEEE 802.11 Rate Adaptation: A Practical Approach”. *ACM Int. Symposium on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM)* (Veneza, out. 2004).

LAKHINA, A.; CROVELLA, M.; DIOT, C. “Diagnosing Network-Wide Traffic Anomalies”. *Proc. 2004 ACM SIGCOMM.*

LAKHINA, A.; CROVELLA, M.; DIOT, C. “Mining Anomalies Using Traffic Feature Distributions”. *Proc. 2005 ACM SIGCOMM.*

LAKSHMAN, T. V.; MADHOW, U. “The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss”. *IEEE/ACM Transactions on Networking,* v.5, n.3 (1997), p. 336–350.

LAM, S. “A Carrier Sense Multiple Access Protocol for Local Networks”. *Computer Networks,* v.4 (1980), p. 21–32.

LARMOUTH, J. *Understanding OSI,* International Thomson Computer Press 1996. O Capítulo 8 deste livro trata de ASN.1 e está disponível on-line em <http://www.salford.ac.uk/iti/books/osi/all.html#head8>.

LARMOUTH, J. *Understanding OSI,* <http://www.business.salford.ac.uk/legacy/isi/books/osi/osi.html>.

LAWTON, G. “Is IPv6 Finally Gaining Ground?” *IEEE Computer Magazine* (ago. 2001), p. 11–15.

LEBLOND, S.; ZHANG, C.; LEGOUT, A.; ROSS, K. W.; DABBOUS, W. “Exploring the Privacy Limits of Real-Time Communication Applications”. *Proc. 2011 ACM Internet Measurement Conference* (Berlim, 2011).

LEBLOND, S.; ZHANG, C.; LEGOUT, A.; ROSS, K. W.; DABBOUS, W. "I Know Where You and What You Are Sharing: Exploiting P2P Communications to Invade Users Privacy". *Proc. 2011 ACM Internet Measurement Conference* (Berlim).

LEIGHTON, T. "Improving Performance on the Internet". *Communications of the ACM*, v.52, n.2 (fev. 2009), p. 44–51.

LEINER, B.; CERF, V.; CLARK, D.; KAHN, R.; KLEINROCK, L.; LYNCH, D.; POSTEL, J.; ROBERTS, L.; WOOLF, S. "A Brief History of the Internet". <http://www.isoc.org/internet/history/brief.html>.

LEUNG, K.; V.; LI, O. K. "TCP in Wireless Networks: Issues, Approaches, and Challenges". *IEEE Commun.Surveys and Tutorials*, v.8, n.4 (2006), p. 64–79.

LI, L.; ALDERSON, D.; WILLINGER, W.; DOYLE, J. "A First-Principles Approach to Understanding the Internet's Router -Level Topology". *Proc. 2004 ACM SIGCOMM* (Portland, ago. 2004).

LI, J.; GUIDERO, M.; WU, Z.; PURPUS, E.; EHRENKRANZ, T. "BGP Routing Dynamics Revisited". *ACM Computer Communication Review* (abr. 2007).

LIANG, J.; NAOUMOV, N.; ROSS, K. W. "The Index Poisoning Attack in P2P File- Sharing Systems". *Proc. 2006 IEEE INFOCOM* (Barcelona, abr. 2006).

LIN, Y.; CHLAMTAC, I. *Wireless and Mobile Network Architectures*. Nova York: John Wiley and Sons, 2001.

LIOGKAS, N.; NELSON, R.; KOHLER, E.; ZHANG, L. "Exploiting BitTorrent For Fun (But Not Profit)". *6th International Workshop on Peer-to-Peer Systems (IPTPS 2006)*.

LIU, B.; GOECKEL, D.; TOWSLEY, D. "TCP-Cognizant Adaptive Forward Error Correction in Wireless Networks". *Proc. 2002 Global Internet.*

LIU, J.; MATTA, I.; CROVELLA, M. "End-to-End Inference of Loss Nature in a Hybrid Wired/Wireless Environment". *Proc. WiOpt'03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks.*

LIU, Z.; DHUNGEL, P.; WU, D.; ZHANG, C.; ROSS, K. W. "Understanding and Improving Incentives in Private P2P Communities". *ICDCS* (Gênova, 2010).

LOCHER, T.; MOOR, P.; SCHMID, S.; WATTENHOFER, R. "Free Riding in BitTorrent is Cheap". *Proc. ACM HotNets 2006* (Irvine, nov. 2006).

LUI, J.; MISRA, V.; RUBENSTEIN, D. "On the Robustness of Soft State Protocols". *Proc. IEEE Int. Conference on Network Protocols (ICNP '04)*, p. 50–60.

LUOTONEN, A. *Web Proxy Servers*. Englewood Cliffs: Prentice Hall, 1998.

LYNCH, D.; ROSE, M. *Internet System Handbook*. Reading: Addison-Wesley, 1993.

MACEDONIA, M.; BRUTZMAN, D. "MBone Provides Audio and Video Across the Internet". *IEEE Computer Magazine*. v.27, n.4 (abr. 1994), p. 30–36.

MAHDAVI, J.; FLOYD, S. "TCP-Friendly Unicast Rate-Based Flow Control". Nota não publicada (jan.1997).

MALWARE 2006. Computer Economics, "2005 Malware Report: The Impact of Malicious Code Attacks". <http://www.computereconomics.com>.

MANET 2012. IETF Mobile Ad-hoc Networks Working Group. <http://www.ietf.org/html.charters/manet-charter.html>.

MAO, Z. M.; CRANOR, C.; BOUDLIS, F.; RABINOVICH, M.; SPATSCHECK, O.; WANG, J. "A Precise and Efficient Evaluation of the Proximity Between Web Clients and Their Local DNS Servers". *Proc. 2002 USENIX ATC.*

MAXMIND 2012. <http://www.maxmind.com/app/ip-location>.

MAYMOUNKOV, P.; MAZIÈRES, D. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". *Proceedings of the 1st International Workshop on Peerto-Peer Systems (IPTPS '02)* (mar. 2002), p. 53–65.

MCKEOWN, N.; IZZARD, M.; MEKKITTIKUL, A.; ELLERSICK, W.; HOROWITZ, M. "The Tiny Tera: A Packet Switch Core". *IEEE Micro Magazine* (jan.–fev. 1997a).

MCKEOWN, N. "A Fast Switched Backplane for a Gigabit Switched Router". *Business Communications Review*, v.27, n.12. <http://tiny-tera.stanford.edu/~nickm/papers/cisco_fasts_wp.pdf>. (1997b)

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. "OpenFlow: Enabling Innovation in Campus Networks". *ACM SIGCOMM Computer Communication Review*, v.38, n.2 (abr. 2008).

MCQUILLAN, J.; RICHER, I.; ROSEN, E. "The New Routing Algorithm for the Arpanet". *IEEE Transactions on Communications*, v.28, n.5 (maio 1980), p. 711–719.

MEDHI, D.; TIPPER D. (eds.). "Special Issue: Fault Management in Communication Networks". *Journal of Network and Systems Management*, v.5. n.2 (jun.1997).

METCALFE, R. M.; BOGGS, D. R. "Ethernet: Distributed Packet Switching for Local Computer Networks". *Communications of the Association for Computing Machinery*, v.19, n.7 (jul. 1976), p. 395–404.

MEYERS, A.; NG, T.; ZHANG, H. "Rethinking the Service Model: Scaling Ethernet to a Million Nodes". *ACM Hotnets Conference*, 2004.

MFA FORUM 2012. IP/MPLS Forum homepage, <http://www.ipmplsforum.org/>.

MIRKOVIC, J.; DIETRICH, S.; DITTRICH, D.; REIHER, P. *Internet Denial of Service: Attack and Defense Mechanisms*. Prentice Hall, 2005.

MOCKAPETRIS, P. V.; DUNLAP, K. J. "Development of the Domain Name System". *Proc. 1988 ACM SIGCOMM* (Stanford, ago. 1988).

MOCKAPETRIS, P. Sigcomm Award Lecture. Vídeo disponível em <http://www.postel.org/sigcomm>.

MOGUL, J. "TCP offload is a dumb idea whose time has come". *Proc. HotOS IX: The 9th Workshop on Hot Topics in Operating Systems* (2003), USENIX Association.

MOLINARO-FERNANDEZ, P.; MCKEOWN, N.; ZHANG, H. "Is IP Going to Take Over the World (of Communications)?". *Proc. 2002 ACM Hotnets.*

MOLLE, M. L.; SOHRABY, K.; VENETSANOPOULOS, A. N. "Space-Time Models of Asynchronous CSMA Protocols for Local Area Networks". *IEEE Journal on Selected Areas in Communications*, v.5, n.6 (1987), p. 956–968.

MOORE, D.; VOELKER, G.; SAVAGE, S. "Inferring Internet Denial of Service Activity". *Proc. 2001 USENIX Security Symposium* (Washington, ago. 2001).

MOORE, D.; PAXSON, V.; SAVAGE, S.; SHANNON, C.; STANIFORD, S.; WEAVER, N. "Inside the Slammer Worm". *2003 IEEE Security and Privacy Conference.*

MOSHCHUK, A.; BRAGIN, T.; GRIBBLE, S.; LEVY, H. "A Crawler-based Study of Spyware on the Web". *Proc. 13th Annual Network and Distributed Systems Security Symposium* (*NDSS 2006*) (San Diego, fev. 2006).

MOTOROLA 2007. "Long Term Evolution (LTE): A Technical Overview". <http://www.motorola.com/staticfiles/Business/Solutions/Industry%20Solutions/Service%20P roviders/Wireless%20Operators/LTE/_Document/Static%20 Files/6834_MotDoc_New.pdf>.

MOULY, M.; PAUTET, M. *The GSM System for Mobile Communications*, Cell and Sys, Palaiseau, França, 1992.

MOY, J. *OSPF: Anatomy of An Internet Routing Protocol*. Reading: Addison-Wesley, 1998.

MUDIGONDA, J.; YALAGANDULA, P.; MOGUL, J. C.; STIEKES, B.; POUFFARY, Y. "NetLord: A Scalable Multi-Tenant Network Architecture for Virtualized Datacenters". *Proc. 2011 ACM SIGCOMM.*

MUKHERJEE, B. *Optical Communication Networks*. McGraw-Hill, 1997.

MUKHERJEE, B. *Optical WDM Networks*. Springer, 2006.

MYSORE, R. N.; PAMBORIS, A.; FARRINGTON, N.; HUANG, N.; MIRI, P.; RADHAKRISHNAN, S.; SUBRAMANYA, V.; VAHDAT, A. "PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric". *Proc. 2009 ACM SIGCOMM.*

NADEL, B. "4G shootout: Verizon LTE vs. Sprint WiMax". *Computerworld*, 3 fev. 2011.

NAHUM, E.; BARZILAI, T.; KANDLUR, D. "Performance Issues in WWW Servers". *IEEE/ACM Transactions on Networking*, v.10, n.1 (fev. 2002).

NAOUMOV, N.; ROSS, K. W. "Exploiting P2P Systems for DDoS Attacks". *Intl Workshop on Peer-to-Peer Information Management* (Hong Kong, maio 2006).

NEGLIA, G.; REINA, G.; ZHANG, H.; TOWSLEY, D.; VENKATARAMANI, A.; DANAHER, J. "Availability in BitTorrent Systems". *Proc. 2007 IEEE INFOCOM*.

NEUMANN, R. "Internet Routing Black Hole". *The Risks Digest: Forum on Risks to the Public in Computers and Related Systems*, v.19, n.12 (maio 1997). <http://catless.ncl.ac.uk/Risks/19.12.html#subj1.1>.

NEVILLE-NEIL, G. "Whither Sockets?" *Communications of the ACM*, v.52, n.6 (jun. 2009), p. 51–55.

NICHOLSON, A.; CHAWATHE, Y.; CHEN, M.; NOBLE, B.; WETHERALL, D. "Improved Access Point Selection". *Proc. 2006 ACM Mobisys Conference* (Uppsala, 2006).

NIELSEN, H. F.; GETTYS, J.; BAIRD-SMITH, A.; PRUD'HOMMEAUX, E.; LIE, H. W.; LILLEY, C. "Network Performance Effects of HTTP/1.1, CSS1, and PNG". *W3C Document*, 1997 (veja também em *Proc. 1997 ACM SIGCOM* (Cannes, set. 1997), p. 155–166.

NIST 2001. National Institute of Standards and Technology, "Advanced Encryption Standard (AES)". Federal Information Processing Standards 197, nov. 2001, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

NIST IPv6 2012. National Institute of Standards, "Estimating IPv6 & DNSSEC Deployment SnapShots". <http://usgv6-deploymon.antd.nist.gov/snap-all.html>.

NMAP 2012. Nmap homepage. <http://www.insecure.com/nmap>.

NONNENMACHER, J.; BIERSAK, E.; TOWSLEY, D. "Parity-Based Loss Recovery for Reliable Multicast Transmission". *IEEE/ACM Transactions on Networking*, v.6, n.4 (ago. 1998), p. 349–361.

NTIA 1998. National Telecommunications and Information Administration (NTIA), US Department of Commerce, "Management of Internet names and addresses". Docket Number: 980212036-8146-02. <http://www.ntia.doc.gov/ntiahome/domainname/6_5_98dns.htm>.

O'DELL, M. "Network Front-End Processors, Yet Again". *Communications of the ACM*, v.52, n.6 (jun. 2009), p. 46–50.

OID Repository 2012. <http://www.oid-info.com/>.

OSI 2012. <http://www.iso.org/iso/en/ISOOnline.frontpage>.

OSS 2012. OSS Nokalva, "ASN.1 Resources". <http://www.oss.com/asn1/>.

PADHYE, J.; FIROIU, V.; TOWSLEY, D.; KUROSE, J. "Modeling TCP Reno Performance: A Simple Model and its Empirical Validation". *IEEE/ACM Transactions on Networking*, v.8 n.2 (abr. 2000), p. 133–145.

PADHYE, J.; FLOYD, S. "On Inferring TCP Behavior". *Proc. 2001 ACM SIGCOMM* (San Diego, ago. 2001).

PAN, P.; SCHULZRINNE, H. "Staged Refresh Timers for RSVP". *Proc. 2nd Global Internet Conference* (Phoenix, dez. 1997).

PAREKH, A.; GALLAGHER, R. "A generalized processor sharing approach to flow control in integrated services networks: the single-node case". *IEEE/ACM Transactions on Networking*, v.1, n.3 (jun.1993), p. 344–357.

PARTRIDGE, C.; PINK, S. "An Implementation of the Revised Internet Stream Protocol (ST-2)". *Journal of Internetworking: Research and Experience*, v.3, n.1 (mar. 1992).

PARTRIDGE, C. et al. "A Fifty Gigabit per second IP Router". *IEEE/ACM Transactions on Networking*, v.6, n.3 (jun.1998), p. 237–248.

PATHAK, A.; WANG, Y. A.; HUANG, C.; GREENBERG, A.; HU, Y. C.; LI, J.; ROSS, K. W. "Measuring and Evaluating TCP Splitting for Cloud Services". *Passive and Active Measurement (PAM) Conference* (Zurich, 2010).

PAXSON, V. "End-to-End Internet Packet Dynamics". *Proc. 1997 ACM SIGCOMM* (Cannes, set. 1997).

PERKINS, A. "Networking with Bob Metcalfe". *The Red Herring Magazine* (nov. 1994).

PERKINS, C.; HODSON, O.; HARDMAN, V. "A Survey of Packet Loss Recovery Techniques for Streaming Audio". *IEEE Network Magazine* (set./out. 1998), p. 40–47.

PERKINS, C. *Mobile IP: Design Principles and Practice*. Reading: Addison-Wesley, 1998.

PERKINS, C. *Ad Hoc Networking*. Reading: Addison-Wesley, 2000.

PERLMAN, R. *Interconnections: Bridges, Routers, Switches, and Internetworking Protocols.* 2. ed. Reading: Addison-Wesley Professional Computing Series, 1999.

PGPI 2012. <http://www.pgpi.org>.

PHIFER, L. "The Trouble with NAT". *The Internet Protocol Journal,* v.3, n.4 (dez. 2000). <http://www.cisco.com/warp/public/759/ipj_3-4/ipj_3-4_nat.html>.

PIATEK, M.; ISDAL, T.; ANDERSON, T.; KRISHNAMURTHY, A.; VENKATARAMANI, A. "Do Incentives Build Robustness in Bittorrent?". *Proc. NSDI* (2007).

PIATEK, M.; ISDAL, T.; KRISHNAMURTHY, A.; ANDERSON, T. "One hop Reputations for Peer-to-peer File Sharing Workloads". *Proc. NSDI* (2008).

PICKHOLTZ, R.; SCHILLING, D.; MILSTEIN, L. "Theory of Spread Spectrum Communication — a Tutorial". *IEEE Transactions on Communications,* v.30, n.5 (maio 1982), p. 855–884.

PINGPLOTTER 2012. PingPlotter homepage, <http://www.pingplotter.com>.

PISCATELLO, D.; LYMAN CHAPIN, A. *Open Systems Networking.* Reading: Addison-Wesley, 1993.

POINT TOPIC 2006. Point Topic Ltd., *World Broadband Statistics Q1 2006.* <http://www.pointtopic.com>.

POTAROO 2012. "Growth of the BGP Table–1994 to Present". <http://bgp.potaroo.net/>.

PPLIVE 2012. PPLive homepage. <http://www.pplive.com>.

QUAGGA 2012. "Quagga Routing Suite". <http://www.quagga.net/>.

QUITTNER, J.; SLATALLA, M. *Speeding the Net: The Inside Story of Netscape and How it Challenged Microsoft.* Atlantic Monthly Press, 1998.

QUOVA 2012. <www.quova.com>.

RAICIU, C.; BARRE, S.; PLUNTKE, C.; GREENHALGH, A.; WISCHIK, D.; HANDLEY, M. "Improving Datacenter Performance and Robustness with Multipath TCP". *Proc. 2011 ACM SIGCOMM.*

RAMAKRISHNAN, K. K.; JAIN, R. "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks". *ACM Transactions on Computer Systems,* v.8, n.2 (maio 1990), p. 158–181.

RAMAN, S.; MCCANNE, S. "A Model, Analysis, and Protocol Framework for Soft State-based Communication". *Proc. 1999 ACM SIGCOMM* (Boston, ago. 1999).

RAMAN, B.; CHEBROLU, K. "Experiences in using WiFi for Rural Internet in India". *IEEE Communications Magazine,* Special Issue on New Directions in Networking Technologies in Emerging Economies (jan. 2007).

RAMASWAMI, R.; SIVARAJAN, K.; SASAKI, G. *Optical Networks: A Practical Perspective.* Morgan Kaufman Publishers, 2010.

RAMJEE, R.; KUROSE, J.; TOWSLEY, D.; SCHULZRINNE, H. "Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks". *Proc. 1994 IEEE INFOCOM.*

RAO, K. R.; HWANG, J. J. *Techniques and Standards for Image, Video and Audio Coding.* Englewood Cliffs: Prentice Hall, 1996.

RAO, A. S.; LIM, Y. S.; BARAKAT, C.; LEGOUT, A.; TOWSLEY, D.; DABBOUS, W. "Network Characteristics of Video Streaming Traffic". *Proc. 2011 ACM CoNEXT* (Tóquio).

RAT 2012. Robust Audio Tool, <http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/>.

RATNASAMY, S.; FRANCIS, P.; HANDLEY, M.; KARP, R.; SHENKER, S. "A Scalable Content-Addressable Network". *Proc. 2001 ACM SIGCOMM* (San Diego, ago. 2001).

REN, S.; GUO L.; ZHANG, X. "ASAP: an AS-aware peer-relay protocol for high quality VoIP". *Proc. 2006 IEEE ICDCS* (Lisboa, jul. 2006).

RESCORLA, E. *SSL and TLS: Designing and Building Secure Systems.* Boston: Addison-Wesley, 2001.

RFC 001. CROCKER, S. "Host Software". RFC 001 (the *very first* RFC!).

RFC 768. POSTEL, J. "User Datagram Protocol". RFC 768, ago. 1980.

RFC 789. ROSEN, E. "Vulnerabilities of Network Control Protocols". RFC 789.

RFC 791. POSTEL, J. "Internet Protocol: DARPA Internet Program Protocol Specification". RFC 791, set. 1981.

RFC 792. POSTEL, J. "Internet Control Message Protocol". RFC 792, set. 1981.

RFC 793. POSTEL, J. "Transmission Control Protocol". RFC 793, set. 1981.

RFC 801. POSTEL, J. "NCP/TCP Transition Plan". RFC 801, nov. 1981.

RFC 826. PLUMMER, D. C. "An Ethernet Address Resolution Protocol — or — Converting Network Protocol Addresses to 48 bit Ethernet Address for Transmission on Ethernet Hardware". RFC 826, nov. 1982.

RFC 829. CERF, V. "Packet Satellite Technology Reference Sources". RFC 829, nov. 1982.

RFC 854. POSTEL, J.; REYNOLDS, J. "TELNET Protocol Specification". RFC 854, maio 1993.

RFC 950. MOGUL, J.; POSTEL, J. "Internet Standard Subnetting Procedure". RFC 950, ago. 1985.

RFC 959. POSTEL J.; REYNOLDS, J. "File Transfer Protocol (FTP)". RFC 959, out. 1985.

RFC 977. KANTOR, B.; LAPSLEY, P. "Network News Transfer Protocol". RFC 977, fev. 1986.

RFC 1028. DAVIN, J.; CASE, J. D.; FEDOR, M.; SCHOFFSTALL, M. "A Simple Gateway Monitoring Protocol". RFC 1028, nov. 1987.

RFC 1034. MOCKAPETRIS, P. V. "Domain Names — Concepts and Facilities". RFC 1034, nov. 1987.

RFC 1035 MOCKAPETRIS, P. V. "Domain Names — Implementation and Specification". RFC 1035, nov. 1987.

RFC 1058. HENDRICK, C. L. "Routing Information Protocol". RFC 1058, jun.1988.

RFC 1071. BRADEN, R.; BORMAN D.; PARTRIDGE, C. "Computing The Internet Checksum". RFC 1071, set. 1988.

RFC 1075. WAITZMAN, D.; PARTRIDGE, C.; DEERING, S. "Distance Vector Multicast Routing Protocol". RFC 1075, nov. 1988.

RFC 1112. DEERING, S. "Host Extension for IP Multicasting". RFC 1112, ago. 1989.

RFC 1122. BRADEN, R. "Requirements for Internet Hosts — Communication Layers". RFC 1122, out. 1989.

RFC 1123. BRADEN, R. (ed.). "Requirements for Internet Hosts — Application and Support". RFC-1123, out. 1989.

RFC 1142. ORAN, D. "OSI IS-IS Intra-Domain Routing Protocol". RFC 1142, fev. 1990.

RFC 1190. TOPOLCIC, C. "Experimental Internet Stream Protocol: Version 2 (ST-II)". RFC 1190, out. 1990.

RFC 1191. MOGUL, J.; DEERING, S. "Path MTU Discovery". RFC 1191, nov. 1990.

RFC 1213. MCCLOGHRIE, K.; ROSE, M. T. "Management Information Base for Network Management of TCP/IP-based internets: MIB-II". RFC 1213, mar. 1991.

RFC 1256. DEERING, S. "ICMP Router Discovery Messages". RFC 1256, set. 1991.

RFC 1320. RIVEST, R. "The MD4 Message-Digest Algorithm". RFC 1320, abr. 1992.

RFC 1321. RIVEST, R. "The MD5 Message-Digest Algorithm". RFC 1321, abr. 1992.

RFC 1323. JACOBSON, V.; BRADEN, S.; BORMAN, D. "TCP Extensions for High Performance". RFC 1323, maio 1992.

RFC 1422. KENT, S. "Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management". RFC 1422.

RFC 1546. PARTRIDGE, C.; MENDEZ, T.; MILLIKEN, W. "Host Anycasting Service". RFC 1546, 1993.

RFC 1547. PERKINS, D. "Requirements for an Internet Standard Point-to-Point Protocol". RFC 1547, dez. 1993.

RFC 1584. MOY, J. "Multicast Extensions to OSPF". RFC 1584, mar. 1994.

RFC 1633. BRADEN, R.; CLARK, D.; SHENKER, S. "Integrated Services in the Internet Architecture: an Overview". RFC 1633, jun. 1994.

RFC 1636. BRADEN, R.; CLARK, D.; CROCKER, S.; HUITEMA, C. "Report of IAB Workshop on Security in the Internet Architecture". RFC 1636, nov. 1994.

RFC 1661. SIMPSON, W. (ed.), "The Point-to-Point Protocol (PPP)". RFC 1661, jul. 1994.

RFC 1662. SIMPSON, W. (ed.), "PPP in HDLC-Like Framing". RFC 1662, jul. 1994.

RFC 1700. REYNOLDS, J.; POSTEL, J. "Assigned Numbers". RFC 1700, out. 1994.

RFC 1752. BRADNER, S.; MANKIN, A. "The Recommendations for the IP Next Generation Protocol". RFC 1752, jan. 1995.

RFC 1918. REKHTER, Y.; MOSKOWITZ, B.; KARRENBERG, D.; DE GROOT, G. J.; LEAR, E. "Address Allocation for Private Internets". RFC 1918, fev. 1996.

RFC 1930. HAWKINSON, J.; BATES, T. "Guidelines for Creation, Selection, and Registration of an Autonomous System (AS)". RFC 1930, mar. 1996.

RFC 1938. HALLER, N.; METZ, C. "A One-Time Password System". RFC 1938, maio 1996.

RFC 1939. MYERS J.; ROSE, M. "Post Office Protocol—Version 3". RFC 1939, maio 1996.

RFC 1945. BERNERS-LEE, T.; FIELDING, R.; FRYSTYK, H. "Hypertext Transfer Protocol — HTTP/1.0". RFC 1945, maio 1996.

RFC 2003. PERKINS, C. "IP Encapsulation within IP". RFC 2003, out. 1996.

RFC 2004. PERKINS, C. "Minimal Encapsulation within IP". RFC 2004, out. 1996.

RFC 2018. MATHIS, M.; MAHDAVI, J.; FLOYD, S.; ROMANOW, A. "TCP Selective Acknowledgment Options". RFC 2018, out. 1996.

RFC 2050. HUBBARD, K.; KOSTERS, M.; CONRAD, D.; KARRENBERG, J.; POSTEL, D. "Internet Registry IP Allocation Guidelines". RFC 2050, nov. 1996.

RFC 2104. KRAWCZYK, H.; BELLARE, M.; CANETTI, R. "HMAC: Keyed-Hashing for Message Authentication". RFC 2104, fev. 1997.

RFC 2131. DROMS, R. "Dynamic Host Configuration Protocol". RFC 2131, mar. 1997.

RFC 2136. VIXIE, P.; THOMSON, S.; REKHTER, Y.; BOUND, J. "Dynamic Updates in the Domain Name System". RFC 2136, abr. 1997.

RFC 2153. SIMPSON, W. "PPP Vendor Extensions". RFC 2153, maio 1997.

RFC 2205. BRADEN, R.; ZHANG, L.; BERSON, S.; HERZOG, S.; JAMIN, S. "Resource ReSerVation Protocol (RSVP)—Version 1 Functional Specification". RFC 2205, set. 1997.

RFC 2210. WROCLAWSKI, J. "The Use of RSVP with IETF Integrated Services". RFC 2210, set. 1997.

RFC 2211. WROCLAWSKI, J. "Specification of the Controlled-Load Network Element Service". RFC 2211, set. 1997.

RFC 2215. SHENKER, S.;WROCLAWSKI, J. "General Characterization Parameters for Integrated Service Network Elements". RFC 2215, set. 1997.

RFC 2326. SCHULZRINNE, H.; RAO, A.; LANPHIER, R. "Real Time Streaming Protocol (RTSP)". RFC 2326, abr. 1998.

RFC 2328. MOY, J. "OSPF Version 2". RFC 2328, abr. 1998.

RFC 2420. KUMMERT, H. "The PPP Triple-DES Encryption Protocol (3DESE)". RFC 2420, set. 1998.

RFC 2453. MALKIN, G. "RIP Version 2". RFC 2453, nov. 1998.

RFC 2460. DEERING, S.; HINDEN, R. "Internet Protocol, Version 6 (IPv6) Specification". RFC 2460, dez. 1998.

RFC 2475. BLAKE, S.; BLACK, D.; CARLSON, M.; DAVIES, E.; WANG, Z.; WEISS, W. "An Architecture for Differentiated Services". RFC 2475, dez. 1998.

RFC 2578. McCLOGHRIE, K.; PERKINS, D.; SCHOENWAELDER, J. "Structure of Management Information Version 2 (SMIv2)". RFC 2578, abr. 1999.

RFC 2579. MCCLOGHRIE, K.; PERKINS, D.; SCHOENWAELDER, J. "Textual Conventions for SMIv2". RFC 2579, abr. 1999.

RFC 2580. MCCLOGHRIE, K.; PERKINS, D.; SCHOENWAELDER, J. "Conformance Statements for SMIv2". RFC 2580, abr. 1999.

RFC 2597. HEINANEN, J.; BAKER, F.; WEISS, W.; WROCLAWSKI, J. "Assured Forwarding PHB Group". RFC 2597, jun. 1999.

RFC 2616. FIELDING, R.; GETTYS, J.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEACH, P.; BERNERS-LEE, T.; FIELDING, R. "Hypertext Transfer Protocol — HTTP/1.1". RFC 2616, jun. 1999.

RFC 2663. SRISURESH, P.; HOLDREGE, M. "IP Network Address Translator (NAT) Terminology and Considerations". RFC 2663.

RFC 2702. AWDUCHE, D.; MALCOLM, J.; AGOGBUA, J.; O'DELL, M.; MCMANUS, J. "Requirements for Traffic Engineering Over MPLS". RFC 2702, set. 1999.

RFC 2827. FERGUSON, P.; SENIE, D. "Network Ingress Filtering: Defeating Denial of Service Attacks which Employ IP Source Address Spoofing". RFC 2827, maio 2000.

RFC 2865. RIGNEY, C.; WILLENS, S.; RUBENS, A.; SIMPSON, W. "Remote Authentication Dial In User Service (RADIUS)". RFC 2865, jun. 2000.

RFC 2961. BERGER, L.; GAN, D.; SWALLOW, G.; PAN, P.; TOMMASI, F.; MOLENDINI, S. "RSVP Refresh Overhead Reduction Extensions". RFC 2961, abr. 2001.

RFC 3007. WELLINGTON, B. "Secure Domain Name System (DNS) Dynamic Update". RFC 3007, nov. 2000.

RFC 3022. SRISURESH, P.; EGEVANG, K. "Traditional IP Network Address Translator (Traditional NAT)". RFC 3022, jan. 2001.

RFC 3022. SRISURESH, P.; EGEVANG, K. "Traditional IP Network Address Translator (Traditional NAT)". RFC 3022, jan. 2001.

RFC 3031. ROSEN, E.; VISWANATHAN, A.; CALLON, R. "Multiprotocol Label Switching Architecture". RFC 3031, jan. 2001.

RFC 3032. ROSEN, E.; TAPPAN, D.; FEDORKOW, G.; REKHTER, Y.; FARINACCI, D.; LI, T.; CONTA, A. "MPLS Label Stack Encoding". RFC 3032, jan. 2001.

RFC 3052. EDER, M.; NAG, S. "Service Management Architectures Issues and Review". RFC 3052, jan. 2001.

RFC 3139. SANCHEZ, L.; MCCLOGHRIE, K.; SAPERIA, J. "Requirements for Configuration Management of IP-Based Networks". RFC 3139, jun. 2001.

RFC 3168. RAMAKRISHNAN, K.; FLOYD, S.; BLACK, D. "The Addition of Explicit Congestion Notification (ECN) to IP". RFC 3168, set. 2001.

RFC 3209. AWDUCHE, D.; BERGER, L.; GAN, D.; LI, T.; SRINIVASAN, V.; SWALLOW, G. "RSVP-TE: Extensions to RSVP for LSP Tunnels". RFC 3209, dez. 2001.

RFC 3221. HUSTON, G. "Commentary on Inter-Domain Routing in the Internet". RFC 3221, dez. 2001.

RFC 3232. REYNOLDS, J. "Assigned Numbers: RFC 1700 is Replaced by an On-line Database". RFC 3232, jan. 2002.

RFC 3246. DAVIE, B.; CHARNY, A.; BENNET, J. C. R.; BENSON, K.; LE BOUDEC, J. Y.; COURTNEY, W.; DAVARI, S.; FIROIU, V.; STILIADIS, D. "An Expedited Forwarding PHB (Per-Hop Behavior)". RFC 3246, mar. 2002.

RFC 3260. GROSSMAN, D. "New Terminology and Clarifications for Diffserv". RFC 3260, abr. 2002.

RFC 3261. ROSENBERG, J.; SCHULZRINNE, H.; CARMARILLO, G.; JOHNSTON, A.; PETERSON, J.; SPARKS, R.; HANDLEY, M.; SCHOOLER, E. "SIP: Session Initiation Protocol". RFC 3261, jul. 2002.

RFC 3272. BOYLE, J.; GILL, V.; HANNAN, A.; COOPER, D.; AWDUCHE, D.; CHRISTIAN, B.; LAI, W. S. "Overview and Principles of Internet Traffic Engineering". RFC 3272, maio 2002.

RFC 3286. ONG, L.; YOAKUM, J. "An Introduction to the Stream Control Transmission Protocol (SCTP)". RFC 3286, maio 2002.

RFC 3346. BOYLE, J.; GILL, V.; HANNAN, A.; COOPER, D.; AWDUCHE, D.; CHRISTIAN, B.; LAI, W. S. "Applicability Statement for Traffic Engineering with MPLS". RFC 3346, ago. 2002.

RFC 3376. CAIN, B.; DEERING, S.; KOUVELAS, I.; FENNER, B.; THYAGARAJAN, A. "Internet Group Management Protocol, Version 3". RFC 3376, out. 2002.

RFC 3390. ALLMAN, M.; FLOYD, S.; PARTRIDGE, C. "Increasing TCP's Initial Window". RFC 3390, out. 2002.

RFC 3410. CASE, J.; MUNDY, R.; PARTAIN, D. "Introduction and Applicability Statements for Internet Standard Management Framework". RFC 3410, dez. 2002.

RFC 3411. HARRINGTON, D.; PRESUHN, R.; WIJNEN, B. "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks". RFC 3411, dez. 2002.

RFC 3414. BLUMENTHAL U.; WIJNEN, B. "User-based Security Model (USM) for Version 3 of the Simple Network Management Protocol (SNMPv3)". RFC 3414, dez. 2002.

RFC 3415. WIJNEN, B.; PRESUHN, R.; MCCLOGHRIE, K. "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)". RFC 3415, dez. 2002.

RFC 3416. PRESUHN, R.; CASE, J.; MCCLOGHRIE, K.; ROSE, M.; WALDBUSSER, S. "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)". dez. 2002.

RFC 3439. BUSH, R.; MEYER, D. "Some internet architectural guidelines and philosophy". RFC 3439, dez. 2003.

RFC 3447. JONSSON, J.; KALISKI, B. "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1". RFC 3447, fev. 2003.

RFC 3468. ANDERSSON, L.; SWALLOW, G. "The Multiprotocol Label Switching (MPLS) Working Group Decision on MPLS Signaling Protocols". RFC 3468, fev. 2003.

RFC 3469. SHARMA, V., (ed.), HELLSTRAND, F. (ed.), "Framework for Multi-Protocol Label Switching (MPLS)-based Recovery". RFC 3469, fev. 2003. <ftp://ftp.rfc-editor.org/in-notes/rfc3469.txt>.

RFC 3501. CRISPIN, M. "Internet Message Access Protocol — Version 4rev1". RFC 3501, mar. 2003.

RFC 3550. SCHULZRINNE, H.; CASNER, S.; FREDERICK, R.; JACOBSON, V. "RTP: A Transport Protocol for Real-Time Applications". RFC 3550, jul. 2003.

RFC 3569. BHATTACHARYYA, S. (ed.). "An Overview of Source-Specific Multicast (SSM)". RFC 3569, jul. 2003.

RFC 3588. CALHOUN, P.; LOUGHNEY, J.; GUTTMAN, E.; ZORN, G.; ARKKO, J. "Diameter Base Protocol". RFC 3588, set. 2003.

RFC 3618. FENNER, B.; MEYER, D. (ed.). "Multicast Source Discovery Protocol (MSDP)". RFC 3618, out. 2003.

RFC 3649. FLOYD, S. "High Speed TCP for Large Congestion Windows". RFC 3649, dez. 2003.

RFC 3748. ABOBA, B.; BLUNK, L.; VOLLBRECHT, J.; CARLSON, J.; LEVKOWETZ, H. Ed., "Extensible Authentication Protocol (EAP)". RFC 3748, jun.2004.

RFC 3782. FLOYD, S.; HENDERSON, T.; GURTOV, A. "The NewReno Modification to TCP's Fast Recovery Algorithm". RFC 3782, abr. 2004.

RFC 3973. ADAMS, A.; NICHOLAS, J.; SIADAK, W. "Protocol Independent Multicast—Dense Mode (PIM-DM): Protocol Specification (Revised)". RFC 3973, jan. 2005.

RFC 4022. RAGHUNARAYAN, R. (ed.), "Management Information Base for the Transmission Control Protocol (TCP)". RFC 4022, mar. 2005.

RFC 4113. FENNER, B.; FLICK, J. "Management Information Base for the User Datagram Protocol (UDP)". RFC 4113, jun. 2005.

RFC 4213. NORDMARK, E.; GILLIGAN, R. "Basic Transition Mechanisms for IPv6 Hosts and Routers". RFC 4213, out. 2005.

RFC 4271. REKHTER, Y.; LI, T.; HARES, S. Ed., "A Border Gateway Protocol 4 (BGP-4)". RFC 4271, jan. 2006.

RFC 4272. MURPHY, S. "BGP Security Vulnerabilities Analysis". RFC 4274, jan. 2006.

RFC 4274. MEYER, D.; PATEL, K. "BGP-4 Protocol Analysis". RFC 4274, jan. 2006.

RFC 4291. HINDEN, R.; DEERING, S. "IP Version 6 Addressing Architecture". RFC 4291, fev. 2006.

RFC 4293. ROUTHIER, S. (ed.). "Management Information Base for the Internet Protocol (IP)". RFC 4293, abr. 2006.

RFC 4301. KENT, S.; SEO, K. "Security Architecture for the Internet Protocol". RFC 4301, dez. 2005.

RFC 4302. KENT, S. "IP Authentication Header". RFC 4302, dez. 2005.

RFC 4303. KENT, S. "IP Encapsulating Security Payload (ESP)". RFC 4303, dez. 2005.

RFC 4305. EASTLAK D. "Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)". RFC 4305, dez. 2005.

RFC 4340. KOHLER, E.; HANDLEY, M.; FLOYD, S. "Datagram Congestion Control Protocol (DCCP)". RFC 4340, mar. 2006.

RFC 4443. CONTA, A.; DEERING, S.; GUPTA, M. Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification". RFC 4443, mar. 2006.

RFC 4346. DIERKS, T.; RESCORLA, E. "The Transport Layer Security (TLS) Protocol Version 1.1". RFC 4346, abr. 2006.

RFC 4502. WALDBUSSER, S. "Remote Network Monitoring Management Information Base Version 2". RFC 4502, maio 2006.

RFC 4514. ZEILENGA, K. (ed.), "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names". RFC 4514, jun. 2006.

RFC 4601. FENNER, B.; HANDLEY, M.; HOLBROOK, H.; KOUVELAS, I. "Protocol Independent Multicast — Sparse Mode (PIM-SM): Protocol Specification (Revised)". RFC 4601, ago. 2006.

RFC 4607. HOLBROOK, H.; CAIN, B. "Source-Specific Multicast for IP". RFC 4607, ago. 2006.

RFC 4611. MCBRIDE, M.; MEYLOR, J.; MEYER, D. "Multicast Source Discovery Protocol (MSDP) Deployment Scenarios". RFC 4611, ago. 2006.

RFC 4632. FULLER, V.; LI, T. "Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan". RFC 4632, ago. 2006.

RFC 4960. STEWART, R. (ed.), "Stream Control Transmission Protocol". RFC 4960, set. 2007.

RFC 4987. EDDY, W. "TCP SYN Flooding Attacks and Common Mitigations". RFC 4987, ago. 2007.

RFC 5000. RFC (ed.). "Internet Official Protocol Standards". RFC 5000, maio 2008.

RFC 5109. LI, A. (ed.). "RTP Payload Format for Generic Forward Error Correction". RFC 5109, dez. 2007.

RFC 5110. SAVOLA, P. "Overview of the Internet Multicast Routing Architecture". RFC 5110, jan. 2008.

RFC 5216. SIMON, D.; ABOBA, B.; HURST, R. "The EAP-TLS Authentication Protocol". RFC 5216, mar. 2008.

RFC 5218. THALER, D.; ABOBA, B. "What Makes for a Successful Protocol?". RFC 5218, jul. 2008.

RFC 5321. KLENSIN, J. "Simple Mail Transfer Protocol". RFC 5321, out. 2008.

RFC 5322. RESNICK, P. (ed.). "Internet Message Format". RFC 5322, out. 2008.

RFC 5348. FLOYD, S.; HANDLEY, M.; PADHYE, J.; WIDMER, J. "TCP Friendly Rate Control (TFRC): Protocol Specification". RFC 5348, set. 2008.

RFC 5411. ROSENBERG, J. "A Hitchhiker's Guide to the Session Initiation Protocol (SIP)". RFC 5411, fev. 2009.

RFC 5681. ALLMAN, M.; PAXSON, V.; BLANTON, E. "TCP Congestion Control". RFC 5681, set. 2009.

RFC 5944. PERKINS, C. (ed.). "IP Mobility Support for IPv4, Revised". RFC 5944, nov. 2010.

RFC 5996. KAUFMAN, C.; HOFFMAN, P.; NIR, Y.; ERONEN, P. "Internet Key Exchange Protocol Version 2 (IKEv2)". RFC 5996, set. 2010.

RFC 6071. FRANKEL, S.; KRISHNAN, S. "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap". RFC 6071, fev. 2011.

RFC 6265. BARTH, A. "HTTP State Management Mechanism". RFC 6265, abr. 2011.

RFC 6298. PAXSON, V.; ALLMAN, M.; CHU, J.; SARGENT, M. "Computing TCP's Retransmission Timer". RFC 6298, jun. 2011.

RHEE, I. "Error Control Techniques for Interactive Low-Bit Rate Video Transmission over the Internet". *Proc. 1998 ACM SIGCOMM* (Vancouver, ago. 1998).

RIVEST, R.; SHAMIR, A.; ADELMAN, L. "A Method for Obtaining Digital Signatures and Public-key Cryptosystems". *Communications of the ACM*, v.21, n.2 (fev. 1978), p. 120–126.

ROBERTS, L.; MERRIL, T. "Toward a Cooperative Network of Time-Shared Computers". *AFIPS Fall Conference* (out. 1966).

ROBERTS, J. "Internet Traffic, QoS and Pricing". *Proc. 2004 IEEE INFOCOM,* v.92, n.9 (set. 2004), p. 1389–1399.

RODRIGUES, R.; DRUSCHEL, P. "Peer-to-Peer Systems". *Communications of the ACM,* v.53, n.10 (out. 2010), p. 72–82.

ROHDE; SCHWARZ. "UMTS Long Term Evolution (LTE) Technology Introduction". Application Note 1MA111.

ROM, R.; SIDI, M. *Multiple Access Protocols: Performance and Analysis.* Nova York: Springer-Verlag, 1990.

ROOT SERVERS 2012. <http://www.root-servers.org/>.

ROSE, M. *The Simple Book: An Introduction to Internet Management, Revised Second Edition.*, Englewood Cliffs: Prentice Hall, 1996.

ROSS, K. W. *Multiservice Loss Models for Broadband Telecommunication Networks.* Berlim: Springer, 1995.

ROWSTON, A.; DRUSCHEL, P. "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems". *Proc. 2001 IFIP/ACM Middleware* (Heidelberg, 2001).

RSA Fast 2012. "How Fast is RSA?" <http://www.rsa.com/rsalabs/node.asp?id=2215>.

RSA Key 2012. RSA Laboratories. "How large a key should be used in the RSA Crypto system?" <http://www.rsa.com/rsalabs/node.asp?id=2218>.

RUBENSTEIN, D.; KUROSE, J.; TOWSLEY, D. "Real-Time Reliable Multicast Using Proactive Forward Error Correction". *Proceedings of NOSSDAV '98* (Cambridge, jul. 1998).

RUBIN, A. *White-Hat Security Arsenal: Tackling the Threats.* Addison-Wesley, 2001.

RUIZ-SÁNCHEZ, M.; BIERSACK, E.; DABBOUS, W. "Survey and Taxonomy of IP Address Lookup Algorithms". *IEEE Network Magazine,* v.15, n.2 (mar./abr. 2001), p. 8–23.

SALTZER, J.; REED, D.; CLARK, D. "End-to-End Arguments in System Design". *ACM Transactions on Computer Systems (TOCS),* v.2, n.4 (nov. 1984).

SANDVINE 2011. "Global Internet Phenomena Report, Spring 2011". <http://www.sandvine. com/news/global broadband trends.asp, 2011>.

SARDAR, B.; SAHA, D. "A Survey of TCP Enhancements for Last-Hop Wireless Networks". *IEEE Commun.Surveys and Tutorials,* v.8, n.3 (2006), p. 20–34.

SAROIU, S.; GUMMADI, P. K.; GRIBBLE, S. D. "A Measurement Study of Peer-to-Peer File Sharing Systems". *Proc. of Multimedia Computing and Networking (MMCN)* (2002a).

SAROIU, S.; GUMMADI, K. P.; DUNN, R. J.; GRIBBLE S. D.; LEVY, H. M. "An Analysis of Internet Content Delivery Systems". *USENIX OSDI* (2002b).

SAYDAM, T.; MAGEDANZ, T. "From Networks and Network Management into Service and Service Management". *Journal of Networks and System Management,* v.4, n.4 (dez. 1996), p. 345–348.

SCHILLER, J. *Mobile Communications* 2.ed. Addison Wesley, 2003.

SCHNEIER, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley and Sons, 1995.

SCHULZRINNE, H. "A Comprehensive Multimedia Control Architecture for the Internet". *NOSSDAV'97 (Network and Operating System Support for Digital Audio and Video)* (St. Louis, maio 1997).

SCHULZRINNE-RTP 2012. <http://www.cs.columbia.edu/~hgs/rtp>.

SCHULZRINNE-RTSP 2012. <http://www.cs.columbia.edu/~hgs/rtsp>.

SCHULZRINNE-SIP 2012. <http://www.cs.columbia.edu/~hgs/sip>.

SCHWARTZ, M. *Computer-Communication Network Design and Analysis.* Englewood Cliffs: Prentice-Hall, 1997.

SCHWARTZ, M. *Information, Transmission, Modulation, and Noise.* Nova York: McGraw Hill, 1980.

SCHWARTZ, M. "Performance Analysis of the SNA Virtual Route Pacing Control". *IEEE Transactions on Communications,* v.30, n.1 (jan.1982), p. 172–184.

SCOURIAS, J. “Overview of the Global System for Mobile Communications: GSM”. <http://www.privateline.com/PCS/GSM0.html>.

SEGALLER, S. *Nerds 2.0.1, A Brief History of the Internet.* Nova York: TV Books, 1998.

SHACHAM, N.; MCKENNEY, P. “Packet Recovery in High-Speed Networks Using Coding and Buffer Management”. *Proc. 1990 IEEE INFOCOM* (San Francisco, abr. 1990), p. 124–131.

SHAIKH, A.; TEWARI, R.; AGRAWAL, M. “On the Effectiveness of DNS-based Server Selection”. *Proc. 2001 IEEE INFOCOM.*

SHARMA, P.; PERRY, E.; MALPANI, R. “IP Multicast Operational Network management: Design, Challenges, and Experiences”. *IEEE Network Magazine* (mar. 2003), p. 49–55.

SINGH, S. *The Code Book: The Evolution of Secrecy from Mary, Queen of Scotsto Quantum Cryptography.* Doubleday Press, 1999.

SIP SOFTWARE 2012. H. Schulzrinne Software Package site. <http://www.cs.columbia.edu/IRT/software>.

SKOUDIS, E.; ZELTSER, L. *Malware: Fighting Malicious Code.* Prentice Hall, 2004.

SKOUDIS, E.; LISTON, T. *Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses (2nd Edition)*, Prentice Hall, 2006.

SKYPE 2012. <www.skype.com>.

SMIL 2012. W3C Synchronized Multimedia homepage, <http://www.w3.org/AudioVideo>.

SMITH, J. “Fighting Physics: A Tough Battle”. *Communications of the ACM,* v.52, n.7 (jul. 2009), p. 60–65.

SNORT 2012. Sourcefire Inc., Snort homepage, <http://http://www.snort.org/>.

SOLARI, S. J. *Digital Video and Audio Compression.* Nova York: McGraw Hill, 1997.

SOLENSKY, F. “IPv4 Address Lifetime Expectations”. in *IPng: Internet Protocol Next Generation.* (S. Bradner, A. Mankin, ed.), Reading: Addison-Wesley, 1996.

SPRAGINS, J. D. *Telecommunications Protocols and Design.* Reading: Addison-Wesley, 1991.

SRIKANT, R. *The Mathematics of Internet Congestion Control.* Birkhauser, 2004.

SRIPANIDKULCHAI, K.; MAGGS B.; ZHANG, H. “An analysis of live streaming workloads on the Internet”. *Proc. 2004 ACM Internet Measurement Conference* (Taormina) p. 41–54.

STALLINGS, W. *SNMP, SNMP v2, and CMIP The Practical Guide to Network Management Standards.* Reading: Addison--Wesley, 1993.

STALLINGS, W. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2.* Reading: Addison-Wesley, 1999.

STEINDER, M.; SETHI, A. “Increasing robustness of fault localization through analysis of lost, spurious, and positive symptoms”. *Proc. 2002 IEEE INFOCOM.*

STEVENS, W. R. *Unix Network Programming.* Englewood Cliffs: Prentice-Hall.

STEVENS, W. R. *TCP/IP Illustrated, v.1: The Protocols.* Reading: Addison-Wesley, 1994.

STEVENS, W. R. *Unix Network Programming, v.1: Networking APIs-Sockets and XTI,* 2.ed. Englewood Cliffs: Prentice -Hall, 1997.

STEVENS, W. R. *BGP4: Interdomain Routing in the Internet.* Addison-Wesley, 1999.

STOICA, I.; MORRIS, R.; KARGER, D.; KAASHOEK, M. F.; BALAKRISHNAN, H. “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications”. *Proc. 2001 ACM SIGCOMM* (San Diego, ago. 2001).

STONE, J.; GREENWALD, M.; PARTRIDGE, C.; HUGHES, J. “Performance of Checksums and CRC's Over Real Data”. *IEEE/ACM Transactions on Networking,* v.6, n.5 (out. 1998), p. 529–543.

STONE, J.; PARTRIDGE, C. “When Reality and the Checksum Disagree”. *Proc. 2000 ACM SIGCOMM.* (Estocolmo, ago. 2000).

STRAYER, W. T.; DEMPSEY, B.; WEAVER, A. *XTP: The Xpress Transfer Protocol.* Reading: Addison-Wesley, 1992.

STUBBLEFIELD, A.; IOANNIDIS, J.; RUBIN, A. "Using the Fluhrer, Mantin, and Shamir Attack to Break WEP". *Proceedings of 2002 Network and Distributed Systems Security Symposium* (2002), p. 17–22.

SUBRAMANIAN, M. *Network Management: Principles and Practice.* Reading: Addison-Wesley, 2000.

SUBRAMANIAN, L.; AGARWAL, S.; REXFORD, J.; KATZ, R. "Characterizing the Internet Hierarchy from Multiple Vantage Points". *Proc. 2002 IEEE INFOCOM.*

SUNDARESAN, K.; PAPAGIANNAKI, K. "The Need for Cross-layer Information in Access Point Selection". *Proc. 2006 ACM Internet Measurement Conference* (Rio de Janeiro, out. 2006).

SU, A.-J.; CHOFFNES, D.; KUZMANOVIC A.; BUSTAMANTE, F. "Drafting Behind Akamai" *Proc. 2006 ACM SIGCOMM.*

SUH, K.; FIGUEIREDO, D. R.; KUROSE J.; TOWSLEY, D. "Characterizing and detecting relayed traffic: A case study using Skype". *Proc. 2006 IEEE INFOCOM* (Barcelona, abr. 2006).

SUNSHINE, C.; DALAL, Y. "Connection Management in Transport Protocols". *Computer Networks.* North-Holland, Amsterdam, 1978.

TARIQ, M.; ZEITOUN, A.; VALANCIUS, V.; FEAMSTER, N.; AMMAR, M. "Answering What-If Deployment and Configuration Questions with WISE". *Proc. 2008 ACM SIGCOMM* (ago. 2008).

TECHNONLINE 2012. "Protected Wireless Networks". online webcast tutorial, <http://www.techonline.com/community/tech_topic/internet/21752>.

TEIXEIRA, R.; REXFORD, J. "Managing Routing Disruptions in Internet Service Provider Networks". *IEEE Communications Magazine* (mar. 2006).

THALER, D.; RAVISHANKAR, C. "Distributed Center-Location Algorithms". *IEEE Journal on Selected Areas in Communications,* v.15, n.3 (abr. 1997), p. 291–303.

THINK 2012. Technical History of Network Protocols, "Cyclades". <http://www.cs.utexas.edu/users/chris/think/Cyclades/index.shtml>.

TIAN, Y.; DEY, R.; LIU, Y.; ROSS, K. W. "China's Internet: Topology Mapping and Geolocating". *IEEE INFOCOM Mini-Conference 2012* (Orlando, 2012).

TOBAGI, F. "Fast Packet Switch Architectures for Broadband Integrated Networks". *Proc. 1990 IEEE INFOCOM,* v.78, n.1 (jan. 1990), p. 133–167.

TOR 2012. TOR: Anonymity Online, <http://www.torproject.org>.

TORRES, R.; FINAMORE, A.; KIM, J. R.; MUNAFO, M. M.; RAO, S. "Dissecting Video Server Selection Strategies in the YouTube CDN". *Proc. 2011 Int. Conf. on Distributed Computing Systems.*

TURNER, J. S. "Design of a Broadcast packet switching network". *IEEE Transactions on Communications,* v.36, n.6 (jun.1988), p. 734–743.

TURNER, B. (2012) "2G, 3G, 4G Wireless Tutorial". <http://blogs.nmscommunications.com/communications/2008/10/2g-3g-4g-wireless-tutorial.html>.

UPnP FORUM 2012. <http://www.upnp.org/>.

VAN DER BERG, R. "How the 'Net works: an introduction to peering and transit". <http://arstechnica.com/guides/other/peering-and-transit.ars>.

VARGHESE, G.; LAUCK, A. "Hashed and Hierarchical Timing Wheels: Efficient Data Structures for Implementing a Timer Facility". *IEEE/ACM Transactions on Networking,* v.5, n.6 (dez. 1997), p. 824–834.

VASUDEVAN, S.; DIOT, C.; KUROSE, J.; TOWSLEY, D. "Facilitating Access Point Selection in IEEE 802.11 Wireless Networks". *Proc. 2005 ACM Internet Measurement Conference* (San Francisco, out. 2005).

VERIZON FIOS 2012. "Verizon FiOS Internet: FAQ". <http://www22.verizon.com/residential/fiosinternet/faq/faq.htm>.

VERIZON SLA 2012. "Global Latency and Packet Delivery SLA". <http://www. verizonbusiness.com/terms/global_latency_sla.xml>.

VERMA, D. C. *Content Distribution Networks: An Engineering Approach.* John Wiley, 2001.

VILLAMIZAR, C.; SONG. C. "High performance tcp in ansnet". *ACM SIGCOMM Computer Communications Review*, v.24, n.5 (1994), p. 45–60.

VITERBI, A. *CDMA: Principles of Spread Spectrum Communication.* Reading: Addison-Wesley, 1995.

VIXIE, P. "What DNS Is Not". *Communications of the ACM*, v.52, n.12 (dez. 2009), p. 43–47.

W3C 1995. The World Wide Web Consortium, "A Little History of the World Wide Web" (1995), <http://www.w3.org/History.html>.

WAKEMAN, I.; CROWCROFT, J.; WANG, Z.; SIROVICA, D. "Layering Considered Harmful". *IEEE Network* (jan.1992), p. 20–24.

WALDROP, M. "Data Center in a Box". *Scientific American* (jul. 2007).

J. WALKER, "IEEE P802.11 Wireless LANs, Unsafe at Any Key Size; An Analysis of the WEP Encapsulation". out. 2000, http://www.drizzle.com/~aboba/IEEE/0-362.zip.

WALL, D. *Mechanisms for Broadcast and Selective Broadcast,* tese de doutorado, Stanford University, jun. 1980.

WANG, B.; KUROSE, J.; SHENOY, P.; TOWSLEY, D. "Multimedia Streaming via TCP: An Analytic Performance Study". *Proc. 2004 ACM Multimedia Conference* (Nova York, out. 2004).

WANG, B.; KUROSE, J.; SHENOY, P.; TOWSLEY, D. "Multimedia Streaming via TCP: An Analytic Performance Study". *ACM Transactions on Multimedia Computing Communications and Applications* (*TOMCCAP*), v.4, n.2 (abr. 2008), p. 16:1–22.

WANG, G.; ANDERSEN, D. G.; KAMINSKY, M.; PAPAGIANNAKI, K.; NG, T. S. E.; KOZUCH, M.; RYAN, M. "c-Through: Part-time Optics in Data Centers". *Proc. 2010 ACM SIGCOMM.*

WEATHERSPOON, S. "Overview of IEEE 802.11b Security". *Intel Technology Journal* (2.nd Quarter 2000), <http://download.intel.com/technology/itj/q22000/pdf/art_5.pdf>.

WEI, W.; WANG, B.; ZHANG, C.; KUROSE, J.; TOWSLEY, D. "Classification of Access Network Types: Ethernet, Wireless LAN, ADSL, Cable Modem or Dialup?". *Proc. 2005 IEEE INFOCOM* (abr. 2005).

WEI, W.; WANG, B.; ZHANG, C.; KUROSE, J.; TOWSLEY, D. "Inference and Evaluation of Split-Connection Approaches in Cellular Data Networks". *Proc. Active and Passive Measurement Workshop* (Adelaide, mar. 2006).

WEI, D. X.; JIN, C.; LOW, S. H.; HEGDE, S. "FAST TCP: Motivation, Architecture, Algorithms, Performance". *IEEE/ACM Transactions on Networking* (2007).

WEISER, M. "The Computer for the Twenty-First Century". *Scientific American* (set. 1991), p. 94–10. <http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html>.

WHITE, A.; SNOW, K.; MATTHEWS, A.; MONROSE, F. "Hookt on fon-iks: Phonotactic Reconstruction of Encrypted VoIP Conversations". *IEEE Symposium on Security and Privacy*, Oakland, 2011.

WIGLE.NET 2012. <http://www.wigle.net>.

WILLIAMS, R. "A Painless Guide to CRC Error Detection Algorithms". <http://www.ross.net/crc/crcpaper.html>.

WILSON, C.; BALLANI, H.; KARAGIANNIS, T.; ROWSTRON, A. "Better Never than Late: Meeting Deadlines in Datacenter Networks". *Proc. 2011 ACM SIGCOMM.*

WIMAX FORUM 2012. <http://www.wimaxforum.org>.

WIRESHARK 2012. <http://www.wireshark.org>.

WISCHIK, D.; MCKEOWN, N. "Part I: Buffer Sizes for Core Routers". *ACM SIGCOMM Computer Communications Review*, v.35, n.3 (jul. 2005).

WOO, T.; BINDIGNAVLE, R.; SU, S.; LAM, S. "SNP: an interface for secure network programming". *Proc. 1994 Summer USENIX* (Boston, jun. 1994), p. 45–58.

WOOD, L. "Lloyds Satellites Constellations". <http://www.ee.surrey.ac.uk/Personal/L.Wood/constellations/iridium.html>.

WU, J.; MAO, Z. M.; REXFORD, J.; WANG, J. "Finding a Needle in a Haystack: Pinpointing Significant BGP Routing Changes in an IP Network". *Proc. USENIX NSDI* (2005).

XANADU 2012. <http://www.xanadu.com/>.

XIAO, X.; HANNAN, A.; BAILEY, B.; NI, L. "Traffic Engineering with MPLS in the Internet". *IEEE Network* (mar./abr. 2000).

XIE, H.; YANG, Y. R.; KRISHNAMURTHY, A.; LIU, Y.; SILBERSCHATZ, A. "P4P: Provider Portal for Applications". *Proc. 2008 ACM SIGCOMM* (Seattle, ago. 2008).

YANNUZZI, M.; MASIP-BRUIN, X.; BONAVENTURE, O. "Open Issues in Interdomain Routing: A Survey". *IEEE Network Magazine* (nov./dez. 2005).

YAVATKAR, R.; BHAGWAT, N. "Improving End-to-End Performance of TCP over Mobile Internetworks". *Proc. Mobile 94 Workshop on Mobile Computing Systems and Applications* (dez. 1994).

YOUTUBE 2009. YouTube 2009, Google container data center tour, 2009.

YU, H.; KAMINSKY, M.; GIBBONS P. B.; FLAXMAN, A. "SybilGuard: Defending Against Sybil Attacks via Social Networks". *Proc. 2006 ACM SIGCOMM* (Pisa, set. 2006).

ZEGURA, E.; CALVERT, K.; DONAHOO, M. "A Quantitative Comparison of Graph-based Models for Internet Topology". *IEEE/ACM Transactions on Networking*, v.5, n.6, (dez. 1997). Ver também <http://www.cc.gatech.edu/projects/gtim> para um pacote de software que gera redes com uma estrutura transit-stub.

ZHANG, L.; DEERING, S.; ESTRIN, D.; SHENKER, S.; ZAPPALA, D. "RSVP: A New Resource Reservation Protocol". *IEEE Network Magazine*, v.7, n.9 (set. 1993), p. 8–18.

ZHANG, L. "A Retrospective View of NAT". *The IETF Journal*, v.3, Issue 2 (out. 2007).

ZHANG, M.; JOHN, W.; CHEN, C. "Architecture and Download Behavior of Xunlei: A Measurement-Based Study". *Proc. 2010 Int. Conf. on Educational Technology and Computers (ICETC)*.

ZHANG, X.; LIU, J.; LI B.; YUM, T.-S. P. "CoolStreamingDONet/: A Data-driven Overlay Network for Peer-to-Peer Live Media Streaming". *Proc. 2005 IEEE INFOCOM* (Miami, mar. 2005).

ZHANG, X.; XU, Y.; LIU, Y.; GUO, Z.; WANG, Y. "Profiling Skype Video Calls: Rate Control and Video Quality". *IEEE INFOCOM* (mar. 2012).

ZHAO, B. Y.; HUANG, L.; STRIBLING, J. S.; RHEA, C.; JOSEPH, A. D.; KUBIATOWICZ, J. "Tapestry: A Resilient Global-scale Overlay for Service Deployment". *IEEE Journal on Selected Areas in Communications*, v.22, n.1 (jan. 2004).

ZIMMERMAN, H. "OS1 Reference Model-The ISO Model of Architecture for Open Systems Interconnection". *IEEE Transactions on Communications*, v.28, n.4 (abr. 1980), p. 425–432.

ZIMMERMANN, P. "Why do you need PGP?" <http://www.pgpi.org/doc/whypgp/en/>.

ZINK, M.; SUH, K.; GU, Y.; KUROSE, J. "Characteristics of YouTube Network Traffic at a Campus Network — Measurements, Models, and Implications". *Computer Networks*, v.53, n.4 (2009), p. 501–514.

ÍNDICE

A

B

C

D

E

F

G

H

I

J

K

L

M

Q

R

S

T

U

V

W

X

Y

Z

KUROSE | ROSS

Redes de computadores e a internet

uma abordagem top-down

6ª edição

[Computação]

Seguindo o sucesso da abordagem top-down de suas edições anteriores, *Redes de computadores e a Internet* tem como foco camadas de aplicação e interfaces de programação propondo ao leitor uma experiência prática com os conceitos de protocolo e redes de computadores antes de trabalhar com mecanismos de transmissão de informação das camadas inferiores das pilhas de protocolos.

Além de manter essa característica inovadora, esta 6ª edição adota o Python em suas aplicações e explora temas recentes e importantes, como as redes 4G e os serviços em nuvem. É indicada para alunos de graduação em ciências da computação, engenharia da computação, análise e desenvolvimento de sistemas e sistemas de informação.

sv.pearson.com.br
A Sala Virtual oferece, para professores, apresentações em PowerPoint e manual de soluções (em inglês). Para estudantes, há exercícios autocorrigíveis, exercícios interativos, material de aprendizagem interativo (*applets*), tarefas extras de programação em Python e Java (em inglês), Wireshark labs (em inglês) e exemplos para implementação de laboratórios (em inglês).

Este livro também está disponível para compra em formato e-book. Para adquiri-lo, acesse nosso site.

ISBN 978-85-8143-677-7

9 788581 436777

www.pearson.com.br